

**ADAPTIVE FUZZY LOGIC BASED  
FRAMEWORK FOR SOFTWARE  
DEVELOPMENT EFFORT  
PREDICTION**

by

Moshood Omolade Saliu

A Thesis Presented to the  
DEANSHIP OF GRADUATE  
STUDIES

In Partial Fulfillment of the  
Requirements for the degree

MASTER OF SCIENCE  
in

Computer Science

King Fahd University of Petroleum & Minerals  
Dhahran, Saudi Arabia  
April 2003

Copyright © 2003 by Moshood Omolade Saliu

King Fahd University of Petroleum & Minerals  
DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **MOSHOOD OMOLADE SALIU** under the direction of his thesis advisor and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER SCIENCE**.

Thesis Committee

---

Dr. Moataz A. Ahmed (Chairman)

---

Dr. Mohammad Al-Suwaiyel (Member)

---

Dr. Jarallah Al-Ghamdi (Member)

---

Dr. Kanaan Faisal  
Department Chairman

---

Prof. Osama A. Jannadi  
Dean of Graduate Studies

---

Date

## **Dedication**

This thesis is lovingly dedicated to my parents:

Mrs. Habibat Abigail Omowumi Saliu, for all I am started in her arms.

&

Alhaji (Chief) Idris Ologe Saliu, for his fervent love for my education.

## **Acknowledgments**

Coming this far is not in anyway by chance, but the special design of Almighty Allah. He alone deserves all praises for making everything possible. Acknowledgement is due to King Fahd University of Petroleum & Minerals for supporting this research.

My sincere appreciation goes to Dr. Moataz Ahmed, for all assistance, advice, encouragement and invaluable support given as my advisor and most importantly as a brother during my adventure in KFUPM and throughout the period of this research work in particular. Thank you for being such a great mentor. I wish to thank my thesis committee members, Dr. Muhammad H. Alsuwaiyel and Dr. Jarallah AlGhamdi, for their help, support, and contributions.

The cooperation of the department Chairman, Dr. Kanaan Faisal is highly appreciated. I would like to thank Prof. David Rine (George Mason University, Virginia), for his interest in my work, and for providing valuable materials.

I also acknowledge my colleagues and very dear friends. The special loving support from my dear Olanike gave me strength when I thought none existed, I deeply appreciate you.

Finally, I wish to express my gratitude to my family members for being patient with me and for their love for my success.

## Table of Contents

Dedication .....	iii
Acknowledgments .....	iv
Table of Contents .....	v
List of Tables .....	viii
List of Figures .....	x
Thesis Abstract .....	xv
خلاصة الرسالة .....	xvi
INTRODUCTION .....	1
1.1 Cost Estimation .....	1
1.1.1 Algorithmic Models .....	3
1.1.2 Non-Algorithmic Models .....	4
1.2 The COCOMO Model .....	8
1.3 Problem Statement .....	9
1.4 Main Contributions .....	10
1.5 Organization of Thesis .....	11
THE CONSTITUENTS OF SOFT COMPUTING .....	12
2.1 Introduction .....	12
2.2 Soft Computing .....	12
2.3 Fuzzy Logic .....	13
2.3.1 Architecture of a Fuzzy Logic System .....	20
2.3.2 Adaptive Fuzzy Systems .....	28
2.4 Neural Networks .....	28
2.5 Evolutionary Computation .....	32
2.5.1 Genetic Algorithms .....	33
2.5.2 Genetic Programming .....	35
CRITICAL SURVEY OF SOFT COMPUTING BASED EFFORT PREDICTION	
APPROACHES .....	39
3.1 Introduction .....	39

3.2 Classification Attributes.....	39
3.3 Evaluation of Prediction Systems – Sample Rating Scheme .....	46
3.4 Related Work on Prediction Systems Attributes.....	49
3.5 Critical Survey of Existing Soft Computing Based Approaches to Effort Prediction.....	51
3.5.1 Fuzzy Logic Based Approaches.....	51
3.5.2 Neural Networks Based Approaches .....	57
3.5.3 Evolutionary Computation Based Approaches .....	61
3.5.4 Neuro-Fuzzy Logic Based Approaches .....	62
3.5.5 Neuro-Genetic Based Approaches .....	65
3.6 Conclusion on Existing Soft Computing Based Prediction Approaches .....	68
THE RESEARCH APPROACH AND FRAMEWORK.....	70
4.1 Introduction.....	70
4.2 The Problem.....	70
4.3 Research Methodology .....	71
4.3.1 The Intermediate COCOMO Model .....	72
4.3.2 Fuzzy Logic Approach to COCOMO .....	75
4.3.3 The Proposed Framework.....	77
4.3.4 Sensitivity Analysis .....	94
4.3.5 COCOMO II and the Framework .....	99
TRAINING .....	101
5.1 Introduction.....	101
5.2 Training Approaches in Neuro-Fuzzy Systems .....	101
5.2.1 Neuro-Fuzzy Hybridization .....	102
5.2.2 The Generic Fuzzy Perceptron.....	103
5.3 Effort Prediction Framework – The Training Approach.....	107
5.3.1 Fuzzy Reasoning in the Nominal Effort Model Using Fuzzy Perceptron .....	108
5.3.2 Training Algorithms for the Effort Prediction Framework.....	110
EXPERIMENTS AND RESULTS.....	125

6.1 Introduction.....	125
6.2 Experimental Design for Nominal Effort Prediction .....	125
6.3 Cost Drivers Implementation.....	129
6.4 Experimental Results and Discussion .....	131
6.4.1 Evaluating Prediction Accuracy .....	132
6.4.2 Experiment I – Adapting all Input Membership Functions .....	133
6.4.3 Experiment II - Adapting Input MFs yielding Minimum Membership Degree .....	151
6.4.4 Experiment III - Comparing Prediction Accuracy of the two Approaches .....	164
6.4.5 Experiment IV – Validation Using the COCOMO Database .....	182
6.4.6 Experimental V – Validating the Training Algorithms.....	191
CONCLUSION.....	201
7.1 Introduction.....	201
7.2 Summary of Contributions.....	201
7.3 Future Research .....	203
Bibliography .....	206
Index .....	212
Appendix A : DATASETS .....	215
Appendix B : COST DRIVERS MFs .....	221
Appendix C : SAMPLE FIS .....	243
Vita .....	248

## List of Tables

Table 1: Rating scheme for soft computing based prediction systems .....	47
Table 2: Ranking of evaluation criteria according to importance to a project.....	48
Table 3: Evaluation of Musilek <i>et al.</i> [41] approach to effort prediction .....	53
Table 4: Evaluation of Idri and Abran [23] approach to effort prediction.....	55
Table 5: Evaluation of MacDonnel <i>et al.</i> [35] approach to effort prediction .....	57
Table 6: Evaluation of Wittig <i>et al.</i> [74] approach to effort prediction .....	59
Table 7: Evaluation of Burgess <i>et al.</i> [9] approach to effort prediction .....	62
Table 8: Evaluation of Hodgkinson <i>et al.</i> [20] approach to effort prediction.....	65
Table 9: Evaluation of Shukla, K. K. [64] approach to effort prediction .....	67
Table 10: COCOMO mode coefficients and scale factors values.....	73
Table 11: COCOMO Cost Drivers .....	74
Table 12: COCOMO Cost Drivers Ratings .....	89
Table 13: COCOMO Effort Multipliers .....	90
Table 14: An Illustration to mimic the Fuzziness of mode using a sample project .	96
Table 15: Summary of the Experimental Objectives .....	132
Table 16: Summary of Prediction Quality using Shouldered and Normalized Error for five Trained FIS with different number of fuzzy sets for the SIZE input variable, showing PRED(25) on training and testing datasets for Approach I. ....	141
Table 17: Summary of Prediction Quality using Shouldered and Non-normalized Error for five Trained FIS using different number of fuzzy sets for the SIZE input variable, showing PRED(25) on training and testing datasets for Approach I. ....	142
Table 18: Summary of Prediction Quality using Un-shouldered and Normalized Error for five Trained FIS using different number of fuzzy sets for the SIZE input variable, showing PRED(25) on training and testing datasets for Approach I. ....	149



Table 19: Summary of Prediction Quality using Un-shouldered and Non-normalized Error for five Trained FIS using different number of fuzzy sets for the SIZE input variable, showing PRED(25) on training and testing datasets for Approach I. ....	150
Table 20: Summary of Prediction Quality using Shouldered, Normalized Error for five Trained FIS using different number of fuzzy sets for the SIZE input variable, showing PRED(25) on training and testing datasets for Approach II .....	156
Table 21: Summary of Prediction Quality using Shouldered, Non-normalized Error for five Trained FIS using different number of fuzzy sets for the SIZE input variable, showing PRED(25) on training and testing datasets for Approach II.....	157
Table 22: Summary of Prediction Quality using Un-shouldered, Normalized Error for five Trained FIS using different number of fuzzy sets for the SIZE input variable, showing PRED(25) on training and testing datasets for Approach II.....	162
Table 23: Summary of Prediction Quality using Un-shouldered, Non-normalized Error for five Trained FIS using different number of fuzzy sets for the SIZE input variable, showing PRED(25) on training and testing datasets for Approach II. ....	163
Table 24: Comparing Prediction Quality of Shouldered/Un-Shouldered MFs tracks using.....	173
Table 25: Comparing RMSRE of Shouldered/Un-Shouldered MFs tracks using the Validation .....	174
Table 26: Rulebase of an FIS and the disorganized version .....	193
Table 27: Experimental result of the organized and disorganized rulebases tested on the training data and validation data, before and after training for 56 epochs. ....	195

## List of Figures

<i>Number</i>	<i>Page</i>
Figure 1: Fuzzy set (a) and Classical set (b) for the linguistic value "high cost" of software projects .....	17
Figure 2: A Simplified fuzzy system architecture .....	21
Figure 3: Fuzzy Inference using max-min method. ....	23
Figure 4: A Simple Neuron.....	29
Figure 5: A Basic Artificial Neuron.....	29
Figure 6: Single-layer two-input perceptron [49] .....	31
Figure 7: Tree Representation of the equation: $(3.0 * Reuse + LOC/(x \ln a))$ .....	37
Figure 8: Adaptive Fuzzy Logic Based Framework using COCOMO .....	77
Figure 9: Distribution of Size (KDSI) of the COCOMO Database Based on Mode of Development.....	82
Figure 10: Distribution of Size (KDSI) of the entire COCOMO Database of 63 Software Projects .....	83
Figure 11: A Triangular Membership Function.....	84
Figure 12: COCOMO scale factors fuzzification .....	85
Figure 13: COCOMO coefficients fuzzification.....	85
Figure 14: Fuzzy sets for hypothetical software sizes .....	86
Figure 15: Fuzzy sets for hypothetical software efforts.....	86
Figure 16: Sample membership functions for the TIME Cost Driver .....	92
Figure 17: Effect of Cost Driver (TIME) on Effort .....	93
Figure 18: Three-Layer Generic Fuzzy Perceptron [47][45] .....	104
Figure 19: Cost Model Rules Represented using Fuzzy Perceptron Structure .....	109
Figure 20: Divisions of the input variable SIZE into fuzzy regions and the corresponding membership functions.....	113
Figure 21: The membership functions of output variable EFFORT based on the SIZE MFs of Figure 20. ....	114

Figure 22: To move an output value $o$ of a fuzzy system closer to a current target value $t$ the consequent fuzzy sets are moved towards $t$ .....	120
Figure 23: The different Experimental Routes for Nominal Effort Training ..	126
Figure 24: Divisions of the input variable SIZE into fuzzy regions and the corresponding membership functions using Unshouldered MFs.....	128
Figure 25: The Error graph during Training of the shouldered-normalized error FIS of Approach I.....	134
Figure 26: Prediction of effort using the Training dataset for testing of shouldered-normalized error FIS of Approach I.....	136
Figure 27: Prediction of effort using the Validation dataset for testing of shouldered-normalized error FIS of Approach I.....	137
Figure 28: The Error graph during Training of the Un-shouldered-normalized error FIS of Approach I.....	143
Figure 29: The Error graph during Training of the Un-shouldered-normalized error FIS of Approach I after 139 Epochs .....	145
Figure 30: Prediction of effort using the Training dataset for validation of Un-shouldered-normalized error FIS of Approach I.....	146
Figure 31: Prediction of effort using the Testing dataset for validation of Un-shouldered-normalized error FIS of Approach I.....	147
Figure 32: The Error graph during Training of the shouldered-normalized error FIS of Approach II.....	152
Figure 33: Prediction of effort using the Training dataset for testing of shouldered-normalized error FIS of Approach II .....	153
Figure 34: Prediction of effort using the Testing dataset for validation of shouldered-normalized error FIS of Approach II .....	154
Figure 35: The Error graph during training of the un-shouldered-normalized error FIS of Approach II.....	159
Figure 36: Prediction of effort using the Training dataset for validation of Un-shouldered-normalized error FIS of Approach II .....	160

Figure 37: Prediction of effort using the Testing dataset for validation of Un-shouldered-normalized error FIS of Approach II .....	161
Figure 38: The prediction quality of shouldered MFs using normalized and non-normalized error measures of Approach I on validation dataset before and after training.....	165
Figure 39: The prediction quality of Un-shouldered MFs using normalized and non-normalized error measure of Approach I on validation dataset before and after training.....	166
Figure 40: Graph of Prediction Quality of different tracks of Approach I using the validation dataset .....	168
Figure 41: Graph of RMSRE of different tracks of Approach I using the Validation dataset .....	169
Figure 42: The prediction quality of shouldered MFs using normalized and non-normalized error measure of Approach II on validation dataset before and after training.....	170
Figure 43: The prediction quality of Un-shouldered MFs using normalized and non-normalized error measure of Approach II on validation dataset before and after training.....	171
Figure 44: Graph of Prediction Quality of Shouldered/Un-Shouldered MFs tracks using the Validation dataset for normalized and non-normalized errors. .	176
Figure 45: Graph of RMSRE of Shouldered/Un-Shouldered MFs tracks using the Validation dataset for normalized and non-normalized errors.....	179
Figure 46: The prediction quality of Untrained and Trained FIS using different Number of MFs.....	181
Figure 47: The RMSRE of the Untrained and Trained FIS using different Number of MFs.....	182
Figure 48: Nominal Effort Prediction of Trained FIS and COCOMO model on COCOMO database. ....	185
Figure 49: Effort Prediction of Trained FIS and COCOMO model adjusted by effort multipliers on COCOMO database.....	186

Figure 50: Percentage error of the nominal effort predictions obtained from trained FIS and COCOMO model using the COCOMO database.....	187
Figure 51: Percentage error of the adjusted effort predictions obtained from trained FIS and COCOMO model using the COCOMO database with effort multipliers. ....	188
Figure 52: Percentage error of the nominal effort predictions obtained from trained FIS and COCOMO model using the COCOMO database for training and testing.....	191
Figure 53: Percentage error of the predictions obtained using the trained and untrained FIS with distorted rules on TRAINING DATA .....	197
Figure 54: Percentage error of the predictions obtained using the trained and untrained FIS with distorted rules on VALIDATION DATA .....	198
Figure 55: Antecedent MFs for the FIS of DATA Cost driver .....	221
Figure 56: Consequent MFs for the FIS of DATA Cost driver .....	222
Figure 57: Antecedents MFs for the FIS of TURN Cost driver.....	223
Figure 58: Consequent MFs for the FIS of TURN Cost driver.....	224
Figure 59: Antecedent MFs for the FIS of TIME Cost driver .....	225
Figure 60: Consequent MFs for the FIS of TIME Cost driver.....	226
Figure 61: Antecedent MFs for the FIS of STOR Cost driver.....	227
Figure 62: Consequent MFs for the FIS of STOR Cost driver .....	228
Figure 63: Antecedent MFs for the FIS of ACAP Cost driver .....	229
Figure 64: Consequent MFs for the FIS of ACAP Cost driver.....	230
Figure 65: Antecedent MFs for the FIS of PCAP Cost driver .....	231
Figure 66: Consequent MFs for the FIS of PCAP Cost driver .....	232
Figure 67: Antecedent MFs for the FIS of SCED Cost driver.....	233
Figure 68: Consequent MFs for the FIS of SCED Cost driver .....	234
Figure 69: Antecedent MFs for the FIS of VIRT Cost driver.....	235
Figure 70: Consequent MFs for the FIS of VIRT Cost driver .....	236
Figure 71: Antecedent MFs for the FIS of AEXP Cost driver.....	237
Figure 72: Consequent MFs for the FIS of AEXP Cost driver .....	238

Figure 73: Antecedent MFs for the FIS of VEXP Cost driver.....	239
Figure 74: Consequent MFs for the FIS of VEXP Cost driver .....	240
Figure 75: Antecedent MFs for the FIS of LEXP Cost driver .....	241
Figure 76: Consequent MFs for the FIS of LEXP Cost driver .....	242
Figure 77: The Input (Mode and Size) MFs before Training.....	243
Figure 78: The Consequent (Effort) MFs before Training .....	244
Figure 79: The Input (Mode and Size) MFs after Training .....	246
Figure 80: The Consequent (Effort) MFs after Training .....	247

## **Thesis Abstract**

NAME: Moshood Omolade Saliu  
TITLE: ADAPTIVE FUZZY LOGIC BASED FRAMEWORK FOR  
SOFTWARE DEVELOPMENT EFFORT PREDICTION  
MAJOR FIELD: COMPUTER SCIENCE  
DATE OF DEGREE: APRIL 2003

Software development effort prediction is one of the most critical activities in managing software projects. Algorithmic effort prediction models, which have dominated the software engineering community, are limited by their inability to cope with uncertainties and imprecision surrounding software projects early in the development life cycle. More recently, attention has turned to a variety of machine learning methods, and soft computing in particular to predict software development effort. There are evidences that soft computing has been able to address some of the problems associated with previous models. However, there is no common ground for assessing and comparing these soft computing based prediction techniques. This thesis presents an evaluation scheme for soft computing based effort prediction techniques. We present a critical survey of the state-of-the-art application of soft computing in development effort prediction. Based on the survey results, we propose and implement a transparent and adaptive fuzzy logic framework for effort prediction.

## خلاصة الرسالة

الإسم : مشهود أمولادي ساليو  
عنوان الرسالة: إطار عمل مبني على المنطق الغائم (Fuzzy Logic) القابل للتكيف لتنبؤ  
الجهد المبذل لتطوير البرمجيات  
التخصص: علوم الحاسب الآلي.  
تاريخ التخرج : أبريل 2003م.

تعتبر عملية تنبؤ الجهد المبذل لتطوير البرمجيات أحد الأنشطة الحرجة في إدارة مشاريع البرامج. النماذج الخوارزمية لعملية تنبؤ الجهد المبذل والتي كانت تهيمن مجتمعات هندسة البرمجة أصبحت محدودة لعدم قدرتها مجابهة التغيرات المبهمة والمصاحبة بمشاريع البرمجة عند مراحلها الأولية لدورة التطوير. في الآونة الأخيرة ، بدأ التركيز نحو الطرق المختلفة لتعليم الآلة ( machine learning ) و الحوسبة المرنة (soft computing) خاصة لتنبؤ الجهد المبذل لتطوير البرمجيات. هناك عدة أدلة أن الحوسبة المرنة لها القدرة بمعالجة بعض المشكلات المرتبطة بالنماذج الخوارزمية. ولكن لا توجد أرضية مشتركة لتقييم ومقارنة تقنيات التنبؤ للجهد المبذل لتطوير البرمجيات المبنية على الحوسبة المرنة. يقدم هذا البحث خطة لتقييم تقنيات التنبؤ للجهد المبذل لتطوير البرمجيات المبنية على الحوسبة المرنة. ونقدم دراسة نقدية لأحدث التطبيقات المستخدمة في الحوسبة المرنة لتطوير عملية تنبؤ الجهد المبذل. وبناءً على نتائج هذه الدراسة ، سوف نطرح ونطبق إطار عمل مرن دوشافافية مبني على المنطق الغائم لتنبؤ الجهد المبذل.

درجة الماجستير في العلوم

جامعة الملك فهد للبترول والمعادن

الظهران – المملكة العربية السعودية



# *CHAPTER 1*

## **INTRODUCTION**

### **1.1 Cost Estimation**

Software cost estimation refers to the predictions of the likely amount of effort, time, and staffing levels required to build a software system. The most helpful form of effort prediction is the one made at an early stage during a project, working primarily from feasibility and requirements specifications documents [30]. However, estimates at the early stages of the development are the most difficult to obtain, and they are often the least accurate, because very little detail is known about the project and the product at its start.

Software cost and schedule estimation supports the planning and tracking of software projects. Effectively controlling the expensive investment of software development is of paramount importance [34][68]. The need for reliable and accurate cost predictions in software engineering is an ongoing challenge [20], because it allows for considerable financial and strategic planning. In many cases in the course of software development, we are solving a problem that has never been solved before, and this simple fact

brings so much uncertainty into the predictions we can make regarding the project.

Software cost estimation techniques can be broadly classified as algorithmic and non-algorithmic models. Algorithmic models are derived from the statistical analysis of historical project data [61], for example, Constructive Cost Model (COCOMO) [3] and Software Life Cycle Management (SLIM) [53]. Non-algorithmic techniques include Price-to-Win [3], Parkinson [3], expert judgment [3], and machine learning approaches [61]. Machine learning is used to group together a set of techniques that embody some of the facets of human mind [61], for example fuzzy systems, analogy, regression trees, rule induction and neural networks. Among the machine learning approaches, fuzzy systems and neural networks are considered to belong to the soft computing group.

In the sequel, we briefly discuss developments overtime involving the use of both algorithmic and non-algorithmic estimation techniques. Section 1.2 would succinctly introduce COCOMO being one of the pioneering efforts at software cost and effort estimation, and for the important role it serves in our work.

### 1.1.1 Algorithmic Models

Software effort estimation spawned some of the first attempts at rigorous software measurement, so it is the oldest, most mature aspect of software metrics. Boehm was the first researcher to look at software engineering from an economic point of view, coming up with a cost estimation model, COCOMO-81 in 1981, after investigating a large set of data from TRW in the 1970s [10]. Putnam also developed an early model known as SLIM in 1978 [67]. COCOMO, SLIM and Albrect's function point [16] methods (i.e. measures amount of functionality in a system) were all based on linear regression techniques by collecting data from past projects. Both COCOMO and SLIM take size of lines of code (about which least is known very early in the project) as the major input to their models. A survey on these algorithmic models is presented in [69], while Boehm *et al.* [4] presents a survey with wider coverage of algorithmic and other cost estimation approaches.

Software development as a creative process rather than a constructive process requires more than ordinarily making predictions based on past project data. Models based on historical data have limitations, because attributes and relationships used to predict software development effort could change over time, and/or differ for software development environments [67]. Existing models rely on accurate estimate of size of software in terms of line

of code LOC, number of user screen, interfaces, complexity, and so on, at a time when uncertainty surrounds the project the most [58].

Algorithmic models such as COCOMO, have failed to present suitable solutions that takes into consideration technological advancements, and therefore the emphasis has been placed on new research [20]. One possible reason why algorithmic models have not proven to provide such solution is that, they are often unable to capture the complex set of relationships (e.g. the effect of each variable in a model to the overall prediction made using the model) that are evident in many software development environments [61]. They can be successful within a particular type of environment, but not flexible enough to perform well outside their domain. Their inability to handle categorical data (that is, data that are specified by a range of values) and most importantly lack of reasoning capabilities (that is ability to draw conclusions or make judgments based on available data) contributed to the number of studies exploring non-algorithmic methods (e.g. fuzzy logic). The next section discusses some of the non-algorithmic models that are soft computing based.

### **1.1.2 Non-Algorithmic Models**

Newer computation techniques to cost estimation that are non-algorithmic were sought in the 1990's. Researchers particularly have turned their

attention to a set of approaches that are soft computing based. These include artificial neural networks, fuzzy logic models and genetic algorithms.

Artificial neural network is able to generalize from trained data set. Over a known set of training data, a neural-network learning algorithm constructs *rules* that fit the data, and fits previously unseen data in a reasonable manner as well [67]. Some of the very early works indicating that neural networks are highly applicable to cost estimation include those of Venkatachalam [71] and Krishna and Satsangi [32]. Other latest works using neural networks in cost estimation are reported in [7] and [6].

A marriage between neural networks and fuzzy logic, Neurofuzzy, was introduced into cost estimation in [20]. A Neurofuzzy system can take the linguistic attributes of a fuzzy system and combine them with the learning and modeling attributes of a neural network to produce transparent, adaptive systems.

Fuzzy logic with its offerings of a powerful linguistic representation can represent imprecision in inputs and outputs, while providing a more expert-knowledge based approach to model building. A study by Hodgkinson and Garratt claims that estimation by expert judgment was better than all regression based models [20].

One of the major researches into fuzzy logic application to cost estimation is that of MacDonell *et al.* [35]. Their approach, starting from [18], took a total leap from application of fuzzy logic to already existing regression-based model, but an expert knowledge based application of fuzzy logic. This particular research has evolved into the development of a tool, FULSOME, to assist project managers in making predictions.

Attempts have been made to fuzzify some of the existing algorithmic models in order to handle uncertainties and imprecision problems surrounding such models. The first realization of the fuzziness of several aspects of one of the best known [29], most successful and widely used model for cost estimation, COCOMO, was that of Fei and Liu [77], where they introduced fuzzy set theory in their work on f-COCOMO. They observed that an accurate estimate of delivered source instruction (KDSI) cannot be done before starting a project, and it is unreasonable to assign a determinate number for it. Jack Ryder [58] investigated the application of fuzzy modeling techniques to two of the most widely used models for effort prediction; COCOMO and the Function-Points models respectively. Idri and Abran [23] applied fuzzy logic to the cost drivers of intermediate COCOMO model. The application of fuzzy logic to represent the *mode* and *size* as input to COCOMO model was later presented in [41]. They presented a two-stage implementation called simple F-COCOMO model and augmented F-COCOMO model respectively.

Fuzzy Logic has also offered itself as a useful tool to aid other techniques for software cost estimation like analogy. Similarity between projects is often used when estimating software effort by analogy. Various authors have put forward various proposals for means of deriving similarity as input to the estimation process; the *nearest neighbor algorithm* [61] is one such approach. This algorithm cannot handle projects attributes described by categorical variables other than binary valued variables. An alternative approach using fuzzy logic was proposed by Idri *et al.* [25][24] to deal with this limitation.

Evolutionary computation has also recently found its usefulness in software effort estimation. Burgess *et al.*[9] applied genetic programming (GP), an application of GA, to software effort estimation.

Bayesian analysis, now considered as part of the constituents of soft computing, was used by Chulani *et al.* [10] to calibrate the 1998 version of the COCOMO II model to 161 data points. On comparing with the 1997 calibration using multiple regressions, the Bayesian approach was adjudged to perform better and more robust. Bayesian analysis was also used in the calibration of the 2000 version of COCOMO II [4], resulting in a higher predictive accuracy as well.

## 1.2 The COCOMO Model

The Constructive Cost Model (COCOMO) is a regression based software cost estimation model developed by Barry Boehm [3]. The original COCOMO model (COCOMO'81) distinguishes between three basic development modes: *organic*, *semidetached*, and *embedded*. This classification is commonly used throughout the software engineering community [41].

In COCOMO, organic-mode is associated with systems of typically low complexity, developed by relatively small software team in a familiar in-house environment. An embedded-mode software project operates with very tight constraints e.g. real-time systems, while a semidetached-mode is somewhere intermediate between organic and embedded [3].

The COCOMO model estimates Person Months (PM) of effort, and it is given by the formula:

$$\text{Effort} = A \times [\text{Size}]^B \times \varepsilon \quad \text{Equation 1}$$

Where  $A$  (constant) and  $B$  (scaling factor) depends on the mode of software development,  $size$  is the size of software project measured in Thousands of Delivered Source Instruction (KDSI), and  $\varepsilon$  represents effort multipliers resulting from selected rating for cost drivers attribute.



The COCOMO model is a set of three models – basic, intermediate, detailed. The models depend on the stage of software development and the level of information available. The basic version has  $\varepsilon = 0$ , and it is used for quick, early, rough estimates of effort, but its accuracy is limited because of its lack of factors to account for other project attributes. The intermediate and detailed versions include such factors as cost drivers in terms of their aggregate impact on overall project costs. Information about these factors is available later in the development life cycle. The intermediate COCOMO model, the most widely used version [23], is presented in much detail in Chapter 4.

### **1.3 Problem Statement**

Soft computing based techniques have been proposed to model imprecision present in the variables that makes up the COCOMO model and imprecision surrounding software effort prediction in general. However, there is still much uncertainty as to what prediction technique suits which type of prediction problem [59]. Choosing between the various techniques is an arduous decision that requires the support of a well-defined evaluation scheme to rank each prediction technique as it applies to any prediction problem.

On top of that, problems with existing techniques include the absence of transparent models that incorporate expert knowledge, information from historical project data, and adaptability in the same framework for effort prediction. This type of framework would explore most information source available to make predictions that could gain the confidence of project managers.

Our motivation to investigate these problems derives from the benefits aforementioned.

## **1.4 Main Contributions**

The main contributions of this thesis are stated as follows:

- Proposing an evaluation scheme for assessing and comparing soft computing based approaches to effort prediction;
- Conducting a critical survey and assessment of the state-of-the-art application of soft computing in development effort prediction;
- Proposing an adaptive fuzzy logic based framework for effort prediction; and
- Implementation of the proposed framework, which presents an effort prediction engine that can be tailored to different environments as, desired.

The proposed framework implements a more natural reasoning and promising results obtained indicate that the move from regression model to the fuzzy based approach is reasonable.

### **1.5 Organization of Thesis**

This thesis is organized as follows. Chapter 2 discusses the constituents of soft computing. Chapter 3 presents our scheme, composed of a set of attributes, for comparing and evaluating prediction systems. It also presents a comparison between exiting approaches based on our scheme. Chapter 4 presents our research approach and the framework at conceptual level. Chapter 5 presents the training algorithms implemented in the framework. Chapter 6 discusses experimental details, results and discussion. Chapter 7 gives the conclusion and further work.

## ***CHAPTER 2***

### **THE CONSTITUENTS OF SOFT COMPUTING**

#### **2.1 Introduction**

This chapter discusses the constituents of soft computing: Fuzzy Logic, Neural Networks, and Evolutionary Computation (EC). We first succinctly introduce soft computing, and then go ahead to discuss the various constituents.

#### **2.2 Soft Computing**

Soft computing is a consortium of methodologies centering in fuzzy logic (FL), artificial neural networks (ANN) and evolutionary computation (EC). What is important, to mention here, is that these methodologies are complementary and synergistic, rather than competitive. They provide in one form or another flexible information processing capability for handling real life ambiguous situations. Soft computing aims to exploit the tolerance for imprecision, uncertainty, approximate reasoning, and partial truth in order to achieve tractability, robustness and low-cost solutions. The guiding principle is to devise methods of computation that lead to an acceptable

solution at low cost, by seeking for an approximate solution to an imprecisely/precisely formulated problem [38].

### 2.3 Fuzzy Logic

Fuzzy Logic starts with the concept of fuzzy set theory. It is a theory of classes with unsharp boundaries, and considered as an extension of the classical set theory [25] [22]. The membership  $\mu_A(x)$  of an element  $x$  of a classical set  $A$ , as subset of the universe  $X$ , is defined by:

$$\mu_A(x) = \begin{cases} 1 & \text{iff } x \in A \\ 0 & \text{iff } x \notin A \end{cases}$$

That is,  $x$  is a member of set  $A$  ( $\mu_A(x) = 1$ ) or not ( $\mu_A(x) = 0$ ). The classical sets where the membership value is either zero or one are referred to as *crisp* sets; either an element belongs, or it does not. In many classifications, however, it is not quite clear whether  $x$  belongs to a set  $A$  or not.

The concept of *fuzzy* sets introduced by Lotfi Zadeh [54] may be viewed as a generalization of the concept of a classical set (crisp set). Thus, fuzzy set can be defined by changing the usual definition of the characteristic function of a crisp set, to introduce degree of membership.

Fuzzy sets allow partial membership. A fuzzy set  $A$  is defined by giving a reference set  $X$ , called the *universe* and a mapping;

$$\mu_A : X \rightarrow [0,1]$$

called the membership function of the fuzzy set  $A$ .  $\mu_A(x)$ , for  $x \in X$ , is interpreted as the degree of membership of  $x$  in the fuzzy set  $A$  [73]. A membership function is a curve that defines how each point in the input space is mapped to a membership value between 0 and 1. The higher the membership  $x$  has in the fuzzy set  $A$ , the more true that it is that  $x$  is  $A$ .

Suppose, for example, that  $X$  is the set of software development projects costs and  $A$  is the fuzzy set of *High Cost* projects, the membership degree of the cost of any project  $x$  in  $X$ , is the degree (between 0 and 1) to which  $x$  can be categorized as a *High Cost* project. At a cost of 24 person-months, for example, the fuzzy set *High Cost* may have membership value 0.9, i.e.

$\mu_{HighCost}(24)=0.9$  while for 8 person-month the membership value

is  $\mu_{HighCost}(8)=0.2$ . This gives a viable model for the vague categories of natural language.

How important is it to be exact right when a rough answer will do? The most important idea in fuzzy logic and the base on which it is built was stated by Lotfi Zadeh thus:

**“As the complexity of a system increases, it becomes more difficult and eventually impossible to make a precise statement about its behavior, eventually arriving at a point of complexity where the fuzzy logic method born in humans is the only way to get at the problem.”**

(Originally identified and set forth by Lotfi A. Zadeh [66])

Related to importance of imprecision, we also have this;

**“Everything is vague to a degree you do not realize till you have tried to make it precise.”**

(Bertrand Russell [17])

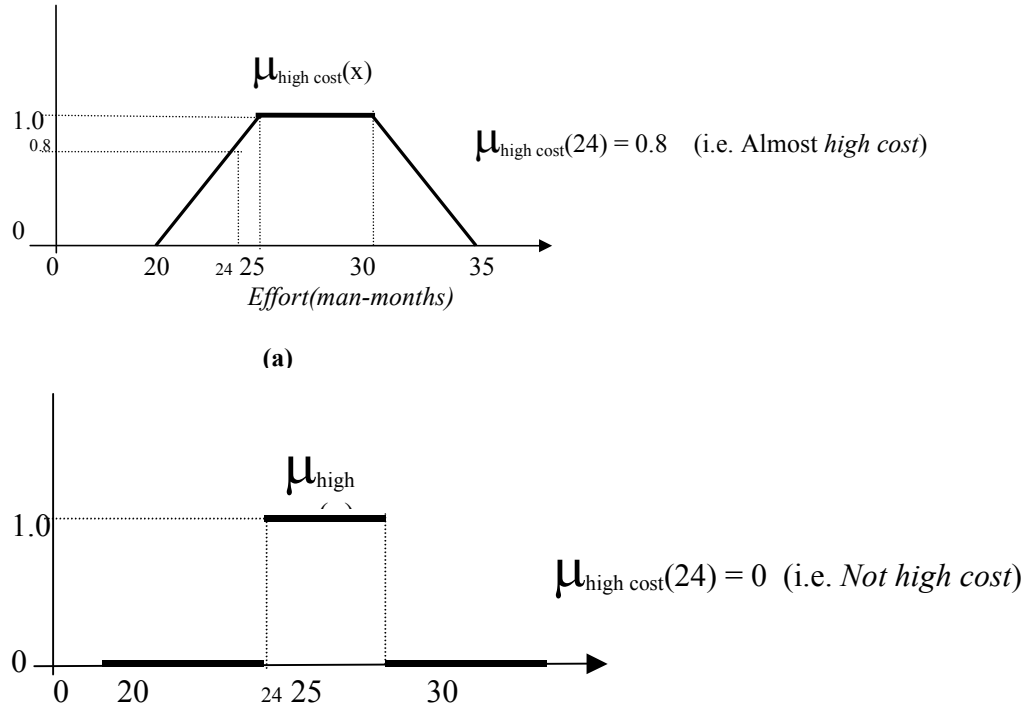
In 1965, Lofti Zadeh formally developed multi-valued set theory, and introduced the term *fuzzy* into the technical literature [50]. Nowadays, the recent emergence of fuzzy commercial products, as well as new theory, has generated a new interest in multi-valued systems. Yet already engineers have successfully applied fuzzy systems in many commercial areas; intelligent subways automation, emergency breakers, cement mixers, control air conditioners, automatic washing machines, guide of robot-arm manipulators,

and so on [50]. Fuzzy systems store banks of fuzzy associations or common sense "rules" [50] such as "*IF size is high, THEN cost is very high*" that might be articulated by a human.

Fuzzy logic is a valuable tool, which can be used to solve highly complex problems where a mathematical model is too difficult or impossible to create. It is also used to reduce the complexity of existing solutions as well as increase the accessibility of control theory [55].

An example of fuzzy sets is defining the set of *high cost* software development project. Suppose projects that cost between 25 and 30 person-months are considered *high cost* projects. In Figure 1 (a), we define a fuzzy set with trapezoidal membership function for *high cost*. In this case, a project that costs 24 man-months is considered *almost high cost* project with membership degree of 0.8 in the set of *high cost* projects. On the other hand, the classical set represented in Figure 1 (b), shows that either a project is considered *high cost* project or not. For instance a project that costs 24 person-months is considered not to belong to the set of *high cost* projects. This shows how fuzzy set helps in capturing or coping with uncertainties and accommodating the use of linguistic variables.





**Figure 1: Fuzzy set (a) and Classical set (b) for the linguistic value "high cost" of software projects**

Fuzzy Logic (FL) was conceived and developed to emulate unclear processes and to use techniques expressed by humans (humans are imprecise by nature). We could think of FL as providing a framework for handling rules (say in control and decision making applications) which have been expressed in an imprecise form. Fuzzy Logic deals with events and situations with subjectively defined attributes. A Proposition in FL does not have to be

either *True* or *False*. An event (or situation) can be, for example, “a bit true”, “fairly true”, “almost true”, “very true” or “not true” depending on the event (or situation) attributes [55].

Logical reasoning in fuzzy logic is a superset of standard Boolean logic. In other words, if the fuzzy values are kept at their extremes of 1 (completely true), and 0 (completely) false, standard logical operations will hold [17]. Therefore, fuzzy logic has operators defined on it in similar fashion to what obtains in the classical two-valued logic. The *AND* operator can be evaluated, for example, using ***min***, i.e. instead of  $A \text{ AND } B$  for intersection, we have  $\min(A, B)$ . Similarly, ***max*** operator replaces the OR and the *NOT* is replaced by  $1-A$  in fuzzy logic.

Thus, the intersection of two fuzzy sets A and B in the universe of discourse X is another fuzzy set in X with the membership function defined for all  $x \in X$  by:

$$\mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x))$$

and for union, we have;

$$\mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x))$$

Just as conditional statements are used in order to make complete sentences, If-Then rules statements are used to formulate the conditional statements that comprise fuzzy logic. A single fuzzy **If-Then Rule** assumes the appearance: **IF (x is A) THEN (y is B)**, where A and B are linguistic values defined by fuzzy sets on the ranges X and Y (universes of discourses). The '**IF**' part is called *antecedent* while '**THEN**' part is called *consequent* or conclusion:

*Antecedent*: Is a condition that returns a single number between 0 and 1.

*Consequent*: Is an assignment that brings up the entire fuzzy set **B** to the output variable y.

Interpreting an If-Then rule implies different parts: first evaluating the antecedent, which involves fuzzifying the input, and apply any *fuzzy operator*, and second, applying that result to the *consequent (implication)*. If the *antecedent* is a fuzzy statement that it is true to some degree of membership, then the *consequent* is also true to that same degree.

Both *antecedent* and *consequent* rules can have multiple parts, e.g.:

*“IF size is high AND complexity is very high THEN cost is very high”*

In this case, all parts of the *antecedent* can be calculated simultaneously and resolved to a single number using the fuzzy AND-operator defined above. However, how does the antecedent affect the consequent? The *consequent* specifies a fuzzy set to be assigned to the output so that the *implication*

*function* must modify that fuzzy set to the degree specified by the *antecedent*.

Such an implication function is discussed in later subsections.

### 2.3.1 Architecture of a Fuzzy Logic System

The architecture of a fuzzy logic system shown in Figure 2 is composed of [54][26]:

1. ***knowledge base or rule base***, expressed in the form of a set of fuzzy rules that express the relation between the input and output fuzzy variables;
2. ***data base*** containing the definitions regarding discretization and normalization of the universes, the definitions of the fuzzy domains of each fuzzy variable and the definition of the membership function for each fuzzy set;
3. ***fuzzifier*** to match numerical input with the fuzzy sets of the domain of the correspondent fuzzy variable;
4. ***fuzzy inference*** mechanism, that is, the algorithm that by firing each of the rules of the knowledge base in parallel obtains an inferred fuzzy set given a set of input values;
5. ***defuzzifier*** algorithm to calculate the crisp output from the inferred fuzzy set;

6. *application area*, for example, software development effort prediction.

The basic fuzzy inference system can take either fuzzy inputs or crisp inputs, but the outputs it produces are usually fuzzy sets that can be defuzzified to extract a crisp output [26]. Figure 2 below also illustrates the data flows between the constituents of a fuzzy based system. For a crisp input, for example, a set of input crisp values relative to variables in the antecedent of the fuzzy rules is introduced; values are fuzzified and the fuzzy inference system is activated. An output fuzzy set is obtained by matching the fuzzified values with the antecedent of all fuzzy rules in the knowledge base (rule base); and finally the output fuzzy set is defuzzified, that is, transformed to a crisp output value.

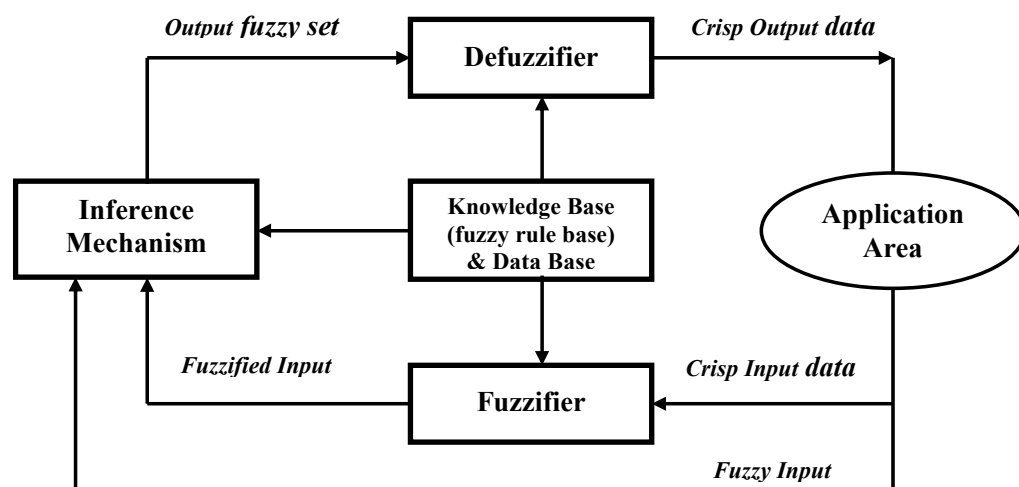


Figure 2: A Simplified fuzzy system architecture

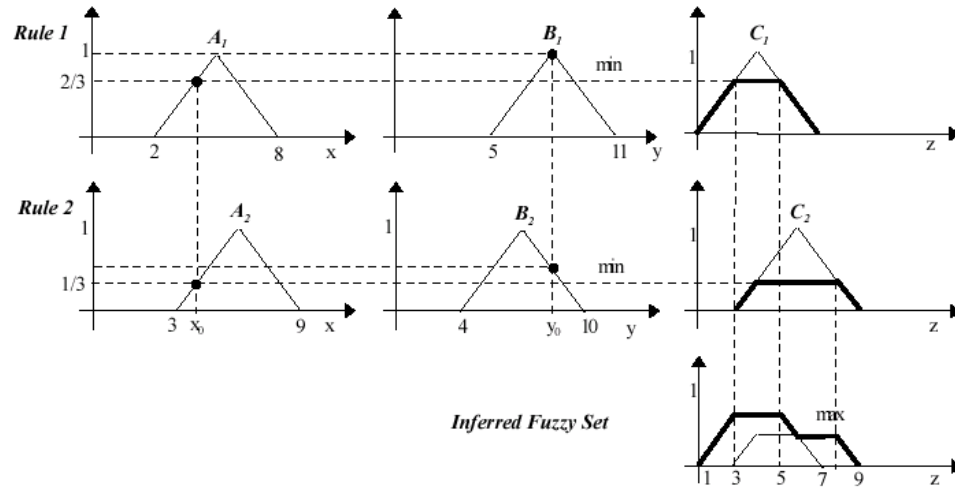
### 2.3.1.1 Fuzzification

The process of matching the input value for a specific fuzzy variable with the fuzzy set that the variable assumes as its value is called *fuzzification*. For more details on fuzzification methods see [76].

### 2.3.1.2 Fuzzy Inference

The *fuzzy inference* mechanism is the process by which the input values for each of the fuzzy variables in the antecedent of the rules ( $x_0$  and  $y_0$  in Figure 3 below) are matched with all rules in the fuzzy rule base and an inferred fuzzy set is obtained. The fuzzy inference is a parallel inference in the sense that all rules contribute in a large or small extent to the inferred result. The weight that a specific rule has on the final output is determined by the degree of matching between the input (fuzzified) values and the rule's antecedent [54].

Figure 3 illustrates a simple fuzzy inference mechanism using the max-min inference introduced by Mamdani *et al.* [76].



**Figure 3: Fuzzy Inference using max-min method.**

**Rule1: IF X is  $A_1$  and Y is  $B_1$  THEN Z is  $C_1$ .**

**Rule2: IF X is  $A_2$  and Y is  $B_2$  THEN Z is  $C_2$**

The inferred fuzzy set can be defuzzified to give a non-fuzzy (crisp) output using any of the defuzzification methods.

### 2.3.1.3 Defuzzification

The process by which a non-fuzzy (crisp) output is obtained from the fuzzy set,  $B^*$  resulting from the fuzzy inference process is called *defuzzification*.

The two most commonly used defuzzification methods are [54]:

#### 1. Center of area (COA) method

This defuzzification method (used for membership function such as triangular and trapezoidal) calculates the center of gravity of the distribution of the degrees of membership of  $B^*$ . If  $B^*$  is defined in the universe X, the crisp output value is obtained for discrete values using

$$z^* = \frac{\sum_{i=1}^q B^*(x_i) \cdot x_i}{\sum_{i=1}^q B^*(x_i)}$$

where  $q$  is the number of quantization levels of universe  $X$ ,  $x_i$  is the crisp value for quantization level  $i$  and  $B^*(x_i)$  is its membership value in the inferred fuzzy set,  $B^*$ . Moreover, for continuous values, we have:

$$z^* = \frac{\int B^*(x) x dx}{\int_x B^*(x) dx}$$

There is also a variant of the COA for monotonic membership functions [54].

## 2. Mean of Maximum (MOM) method

The MOM method (used for any membership function shape) calculates a crisp output value by averaging only the part of the inferred fuzzy set  $B^*$  with maximal degree of membership (equal to the height of fuzzy set  $B^*$ ) [76]. If  $B^*$  is defined in a universe  $X$ , this is calculated for using the formula

$$z^* = \sum_{i=1}^l \frac{x_i}{l}$$



where  $l$  is the number of elements,  $x_i$ , with membership equal to the “height” of fuzzy set  $B^*$ .

The two most commonly used defuzzification methods: COA and MOM methods differ in that [54]:

- MOM selects only the rules which have a strong influence on the inferred result whereas COA considers the weight of every rule to compute the final result; and
- The defuzzified result obtained by MOM is independent of the shape of the membership functions used for the fuzzy sets in the consequent of the rules, as long as the fuzzy sets have a symmetric shape (only one element of the support reaches the maximum membership value). In the case of COA, the shape of the consequent fuzzy sets is important, because the membership value of all elements of the support of the inferred fuzzy set is taken into consideration and not only those for which the membership function achieves the maximum.

Section 2.2.1.4 below compares results from COA and MOM using an illustrative example.

#### 2.3.1.4 A Fuzzy Inference Example

In the following, an example presented in Figure 3 [54] together with its associated fuzzy rule base consisting of the two rules shown in the figure is used to demonstrate the max-min fuzzy inference. We present how to calculate the crisp defuzzified values using the COA and MOM methods.

Suppose  $x_0 = 4$  and  $y_0 = 8$  (see Figure 3) are the input values for fuzzy variables X and Y. First, the inputs  $x_0$  and  $y_0$  have to be matched against fuzzy sets  $A_1$  and  $B_1$ , respectively. This will produce  $A_1(x_0) = \frac{2}{3}$  and  $B_1(y_0) = 1$ . Similarly, for rule 2, we have  $A_2(x_0) = \frac{1}{3}$  and  $B_2(y_0) = \frac{2}{3}$ .

The strength of rule 1 is calculated by

$$a_1 = \text{Min}(A_1(x_0), B_1(y_0)) = \text{Min}(\frac{2}{3}, 1) = \frac{2}{3}$$

similarly for rule 2:

$$a_2 = \text{Min}(A_2(x_0), B_2(y_0)) = \text{Min}(\frac{1}{3}, \frac{2}{3}) = \frac{1}{3}$$

Thus, applying  $a_1$  to the conclusion of rule 1, results in the highlighted trapezoid fuzzy set of Figure 3 for  $C_1$ . Similarly, applying  $a_2$  to the conclusion of rule 2, results in the highlighted trapezoid fuzzy set  $C_2$ .

The membership function for the fuzzy set inferred from rules 1 and 2 is obtained by operating fuzzy sets  $C_1$  and  $C_2$  with the *Max* operator (as shown by the graphic at the bottom of Figure 3).

Using the COA method, the defuzzified output value is:

$$Z_{COA} = \frac{2 \cdot \frac{1}{3} + 3 \cdot \frac{2}{3} + 4 \cdot \frac{2}{3} + 5 \cdot \frac{2}{3} + 6 \cdot \frac{1}{3} + 7 \cdot \frac{1}{3} + 8 \cdot \frac{1}{3}}{\frac{1}{3} + \frac{2}{3} + \frac{2}{3} + \frac{2}{3} + \frac{1}{3} + \frac{1}{3} + \frac{1}{3}} = 4.7$$

Using MOM, three support values (3, 4 and 5) reach the maximum membership value,  $\frac{2}{3}$  in the inferred fuzzy set. Hence, we have

$$Z_{MOM} = \frac{3 + 4 + 5}{3} = 4.0$$

From the sample defuzzification shown above, it can be noticed that MOM concentrates on the support values with maximal membership degree of the output fuzzy set and ignores the contribution of support values with lower membership degrees. The COA output on the other hand considers all support values and therefore provides an answer with a broader range.

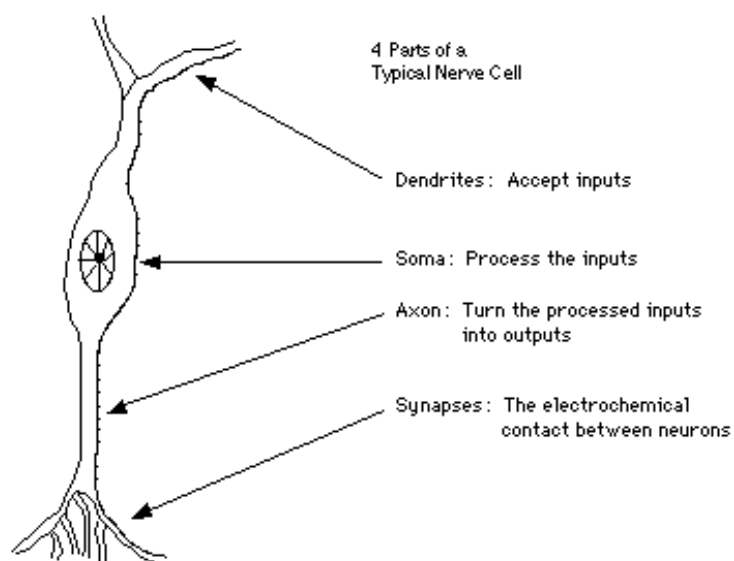
The defuzzification method (COA or MOM) chosen at any point in time depends on the fuzzy logic system application, whether the need is to focus on broad range or concentrated values.

### **2.3.2 Adaptive Fuzzy Systems**

An adaptive fuzzy system is a fuzzy logic system equipped with a training algorithm, where the fuzzy logic system is constructed from a set of fuzzy IF-THEN rules using fuzzy logic principles, and the training algorithm adjusts the parameters of the fuzzy logic system based on feedback information [73]. Adaptive fuzzy systems have their rules automatically generated or updated through training. Thus, an adaptive fuzzy system thinks in a way, since it tunes its rules as new data are supplied.

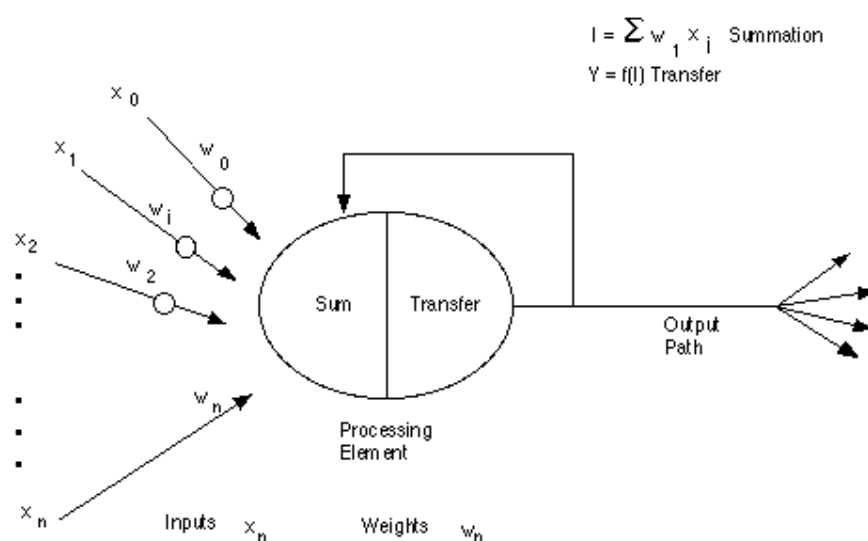
## **2.4 Neural Networks**

Artificial Neural Networks were inspired by a basic knowledge of how the brain works. The brain consists of a densely interconnected set of nerve cells, or basic information-processing units, called *neurons* [49]. A biological neuron receives inputs from other sources, combines them in some way, performs a generally nonlinear operation on the result, and then outputs the result. Figure 4 shows the relationship of these four parts [1].



**Figure 4: A Simple Neuron**

To achieve this, the basic unit of neural networks, the artificial neurons, simulates the four basic functions of natural neurons. Figure 5 shows a fundamental representation of an artificial neuron



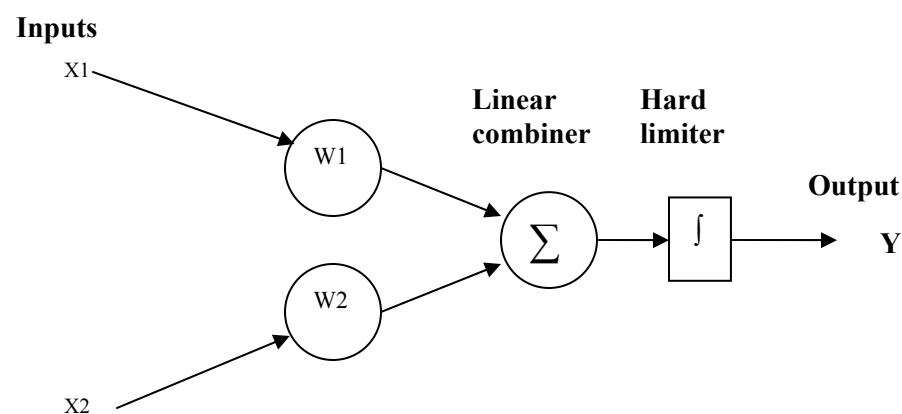
**Figure 5: A Basic Artificial Neuron**

Each neuron is an elementary information-processing unit. To build an artificial neural network (ANN), we must decide first how many neurons are to be used and how the neurons are to be connected to form a network. In other words, we must first choose the network architecture. In Figure 5, various inputs to the neuron are represented by the mathematical symbol,  $x(n)$ . Each of these inputs is multiplied by a connection weight. These weights are represented by  $w(n)$ . In the simplest case, these products are simply summed, fed through a transfer function to generate a result, and then output [1].

In general, input values are multiplied by the weights, summed, passed through a step function and then on to other neurons and finally to the output neuron. The weights are adjusted systematically, based on a given dataset, to optimize the output vector produced from a given input vector. A neural network learns through repeated adjustments of these weights [49], that is training.

In 1958, Frank Rosenblatt introduced a training algorithm that provided the first procedure for training a simple ANN – *The Perceptron*. The perceptron is the simplest form of an ANN. It consists of a single neuron with adjustable weights and a hard limit function (e.g. step and sign functions). A single-layer two-input perceptron is shown in Figure 6. This single-layer perceptron

was able to classify only linearly separable functions, to cope with problems that are not linearly separable, a multilayer neural network is needed. There are different learning algorithms available for training ANN, but the most popular method is the back-propagation learning algorithm discussed by Michael Negnevitsky in [49].



**Figure 6: Single-layer two-input perceptron [49]**

ANNs are recognized for their ability to provide good results when dealing with problems where there are complex relationships between inputs and outputs, and where the input data is distorted by high noise levels. Meanwhile, the performance of neural networks depends on the architecture of the network and their parameter settings. Wittig and Finnie [74] observed that, determining the architecture of a network (size, structure, and

connectivity) affects the performance criteria, such as accuracy of learning, noise resistance, generalization ability etc.

The network of an ANN is in itself a model because the topology and transfer functions of the nodes are usually formulated to match the current problem. Many network architectures have been developed, e.g. the back-propagation networks [74].

While investigating machine learning based prediction systems, Mair *et al.* [37] showed that neural networks offer accurate prediction, but concluded that they are difficult to configure and interpret.

## 2.5 Evolutionary Computation

Evolutionary computation (EC) is based on the computational models of natural selection and genetics. EC works by simulating a population of individuals, evaluating their performance, generating a new population, and repeating this process a number of times. In essence, it simulates evolution by using processes of *selection*, mutation and *reproduction* [49].

EC combines three main techniques: genetic algorithms (GA), evolutionary strategies (ES), and genetic programming (GP). In the sequel, we briefly introduce GA and GP, since they are of more interest in software effort prediction than ES.



### 2.5.1 Genetic Algorithms

Genetic Algorithms (GA) were developed as an alternative technique for tackling general optimization problems with large search spaces. They have the advantage that they do not need any prior knowledge, expertise or logic related to the particular problem being solved [9]. For example, one may want to search for an optimal selection of feature subsets of any software project that predicts effort with minimal error.

Genetic algorithms are based on the Darwinian theory of evolution, which says that genetic operations between chromosomes eventually leads to fitter individuals which are more likely to survive (*survival of the fittest*). Thus, only the most suited elements in a population are likely to survive and generate offspring, and transmit their biological heredity to the new generation [64]. Over a long period, the population of the species as a whole improves.

In the implementation of GAs, usually a solution to the problem being solved is represented by a population of fixed length binary strings, which is termed chromosomes by analogy with the biological equivalent [9]. The GA then iteratively creates new populations from the old by ranking the strings and interbreeding the fittest to create new strings, which are (hopefully) closer to the optimum solution to the problem at hand. GAs use what is termed a *fitness function* in order to select the fittest string that will be used to create

new, and conceivably better, populations of strings [70]. A fitter individual is one that is nearer to the optimal solution. In the context of cost estimation, a fitter solution minimizes the error between the predicted values and the true values.

The basic process of the genetic algorithm is as follows, although a number of variations are possible [9]:

1. Generate at random a population of solutions, i.e., a family of chromosomes.
2. Create a new population from the previous one by applying genetic operators to the fittest chromosomes, or pairs of fittest chromosomes of the previous population.
3. Either repeat step (2), until the fitness of the best solution converged or a specified number of generations have been produced.

The best solution in the final generation is taken as the best approximation to the optimum for that problem that can be attained in that run. The whole process is normally run a number of times, using different seeds to the pseudo-random number generator.

The key parameters that have to be determined for any given problem are [9]:

1. The best way of representing the problem variable domain as a chromosome.
2. The best combination of genetic operators. For GA, reproduction, crossover and mutation are the most common.
3. Choosing the best fitness function, to measure the fitness of a solution.
4. Trying to keep enough diversity in the solutions in a population to allow the process to converge to the global optimum but not converge prematurely to a local optimum.

GP, an application of GA has been used in software effort estimation in the literature (see [14] and [9]). A brief introduction to genetic programming is given in the sequel.

### **2.5.2 Genetic Programming**

Genetic programming (GP) is a variation of GAs, in which the representation of the chromosomes is not restricted to be a binary string but maybe other data structures. Since there is no restriction on the data structure to be evolved by the evolutionary algorithm, the result can be a program, an equation, a circuit, a plan or any other representation [14]. In other words,

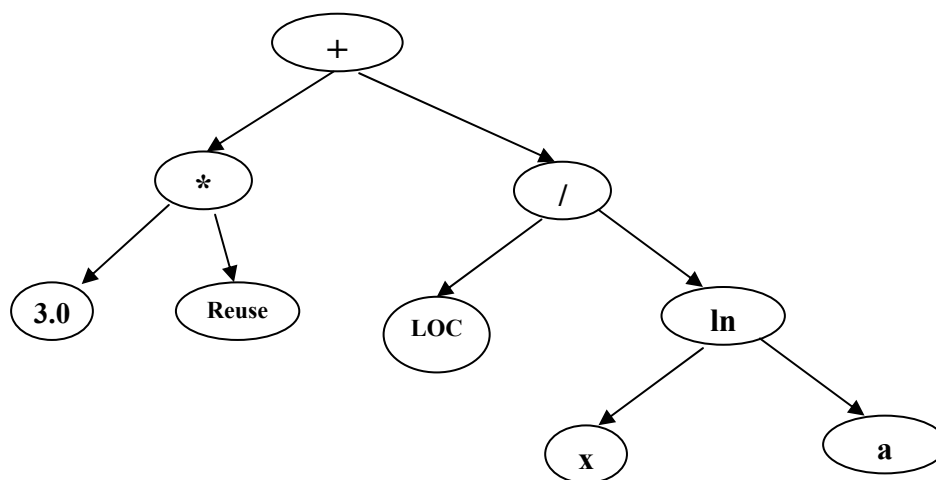
GP can create computer programs as the solution, while GAs create a string of binary numbers that represent the solution [49]. One of the simplest forms of program, which is sufficient for this application, is a binary tree containing operators and operands. This implies that each solution is an algebraic expression [9].

In order to apply GP to effort estimation, for instance, there is a need for effort estimation problem to be formulated as a symbolic regression problem. This implies that, given a number of sets of data values for a set of input parameters and one output parameter, construct an expression of the input parameters that predicts the value of the output parameter for any set of values of the input parameters [9]. The structures to be evolved in the symbolic regression problem are binary trees. The elements to be evolved are trees representing equations. That is, the population  $P$  of the GP algorithm is a set of trees to which the crossover and mutation operators are applied. The terminal nodes are constants or variables, and the non-terminals are the functions which are available for system definition, for example,  $+$ ,  $-$ ,  $*$ ,  $/$ , square root, natural logarithm, exponential etc [14].

The initial preparation for a GP system has several steps. First, it is necessary to choose a suitable alphabet of operands and operators. The operands are normally the independent input variables to the system, while the operators

are expected to be rich enough to cover the type of functionality expected in solutions, e.g. trigonometric function. Second, it is necessary to construct an initial population [9].

Figure 7 below gives an example of a hypothetical cost estimation problem consisting of features like LOC, Reuse, two other variables a and b, and the resulting functionality of the solution



**Figure 7: Tree Representation of the equation:  $(3.0 * \text{Reuse} + \text{LOC} / (x \ln a))$**

The main genetic operations used in GPs are reproduction and crossover. Mutation is rarely used [9]. Reproduction is the copying of one individual from the previous generation into the next generation unchanged. The *crossover* operator chooses a node at random in the first chromosome, called crossover point 1, and the branch to that node of the tree is cut. Then it

chooses a node at random in the second chromosome, called crossover point 2, and the branch to the node is cut. The two sub-trees produced below the cuts are then swapped.

Since trees tend to grow as the algorithm progresses, a maximum depth is normally imposed (or maximum number of nodes) so that the trees do not get too large. Any trees produced by crossover that are too large are discarded. However, the basic algorithm is the same as for GA.

In concluding this chapter, it is worth stating that, the interactivity between the different constituents of soft computing underlies the ability of soft computing to solve complex problems that exceed the capability of individual constituent when used alone. A combination of soft computing methods that has the highest visibility today is that of neuro-fuzzy systems [75]. We are beginning to see systems that are fuzzy-genetic, neuro-genetic, neuro-chaotic and neuro-fuzzy-genetic, among others [75].

## *CHAPTER 3*

### **CRITICAL SURVEY OF SOFT COMPUTING BASED EFFORT PREDICTION APPROACHES**

#### **3.1 Introduction**

In this chapter, we begin in Section 3.2 with the discussion of our proposed classification attributes for soft computing based effort prediction techniques. We present a rating scheme built on the classification attributes to assess and evaluate prediction systems in Section 3.3. Section 3.4 presents related works on prediction system attributes. Section 3.5 discusses a critical survey of existing soft computing based approaches to effort prediction, using our proposed attributes set.

#### **3.2 Classification Attributes**

A software project manager will need, in some sense, to trust predictions made by prediction techniques, if these techniques are to be deployed in practice [30]. Prediction can be viewed from three different perspectives: the prediction problem (e.g. estimating the cost of development/testing, the duration of development/testing etc.), the particular problem at hand (a

specific case with its own characteristics e.g., a supermarket software), and finally the prediction technique (e.g. algorithmic model, analogy, fuzzy logic etc.). Assessing a prediction technique, in our own view, should not only be based on accuracy of estimates made, but also on characteristics of the underlying prediction model. There is still much uncertainty as to what prediction technique suits which type of prediction problem [62]. This type of uncertainties is what we desire to address by proposing set of attributes to aid the assessment and choice of prediction system.

These assessment attributes offer more, beyond their usefulness in carrying out comparison of prediction systems. They can also serve as guidance to researchers attempting to develop prediction systems for effort estimation using soft computing.

To the best of our knowledge, assessing prediction systems in the literature have been based on the characteristics of the dataset (i.e. the data on which the prediction is carried out) and the accuracy of the model only. This fact is further substantiated by the words of Shepperd *et al.* [62] that, “The question which technique or techniques are *best* should be modified to include notion of context. In other words, what is the relationship between different properties of the dataset and the *accuracy* of a prediction system?”



To enable us discuss and assess existing soft computing approaches to effort prediction, and based on our survey of the literature we propose the following general attributes for assessing, classifying, and comparing soft computing based effort prediction systems:

### **1. Underlying Model**

The underlying model specifies whether the soft computing approach to effort prediction is based on an existing algorithmic cost estimation model like COCOMO, SLIM etc or based on other models like expert judgment, analogy etc. Empirical research has indicated that expert judgment coupled with prediction systems out-perform either prediction systems or expert judgment alone [37]. Based on this observation, it might not make much sense to keep everything to a single model. For example, if the underlying model were known not to always perform accurately, this would affect any variant of it that is implemented.

### **2. Trainability**

Trainability is the ability of a prediction system to learn the relationships between features and adapt during training. This factor is what has generally been referred to as adaptability in many of the approaches surveyed.

### **3. Adaptability**

This attribute describes the ability and ease of the prediction system to adjust to new environments as new information and knowledge are supplied. Trainability does not translate to adaptability, as a system could be trainable but not adaptive. Adaptability subsumes trainability.

### **4. Sensitivity**

This attribute refers to the responsiveness to changes in input data, and the type of input data it can handle (e.g. numeric data, categorical data). Responsiveness to changes in input data assesses the effect an imprecision in input to the model has on the effort estimate produced. For example, we desire to know how well the system can accommodate an error involving *size* supplied as 900 KLOC as opposed to the actual 850 KLOC.

### **5. Aspect Coverage**

This refers to the ability of the approach to cover wide range of *aspects* of the development process and environment. For example, whether the effort prediction system takes into consideration the following; *reuse*, *capability-maturity model (CMM) level*, etc.

## **6. Spectrum Coverage**

This attribute refers to the coverage of different types (classes) of systems, e.g. organic, semi-detached and embedded systems, as proposed by Boehm in [3]. If a prediction system is not sophisticated, it might not be able to cover the whole spectrum. A prediction system might still be able to model a software project inherently made up of different classes without necessarily breaking the project into different groups, although such systems may be too complicated.

## **7. Implementation Technique**

We define this attribute to capture the implementation approach taken. An implementation approach using soft computing can be as simple as a straightforward application of an underlying model by applying a single soft computing approach, e.g., fuzzifying input/output, or a more sophisticated implementation technique that explores/combines various capabilities of the soft computing methods used (e.g. fuzzy logic, neural networks, neuro-fuzzy, neuro-genetic etc.)

## **8. Input Data**

This attribute identifies the type of input data required by the effort prediction system to perform estimates, e.g. lines-of-code. It also reflects the ease of getting the input data and the accuracy in making

reasonable estimates. Input data is simply the input required to make estimates using the prediction system, but not to develop the prediction system.

### **9. Knowledge Acquisition and Data Source**

This refers to the mode of knowledge acquisition considered in developing the prediction system, the source of data required and how reliant the system is on the data source. The mode of knowledge acquisition could either be *manual*, with users being the source of the knowledge or *automatic* (based on perceived relationship between the data, through learning). For example, a system that relies on users to supply rules in a fuzzy logic-based approach is said to exhibit manual knowledge acquisition. The data source can be either from historical or simulated data.

### **10. Complexity of the Model**

This attribute refers to the amount of effort or size (e.g. number of neurons, number of rules etc) required for building and/or using the prediction system. This attribute reflects the performance of the system from an efficiency point of view. A model that is not practical or rather difficult to use might not be a good model. For example, neural networks are known to give good approximations, but the question

again boils down to the number and organization of the neurons needed to achieve this, it might be overly complex. Mair *et al.* [37] stated that, “configuring neural nets requires considerable effort and expertise...for this reason it is difficult to see how neural nets could be used within project estimation context by the knowledge engineer”.

### **11. Accuracy**

Accuracy is an attribute of a prediction system that reflects the performance of the model from an effectiveness point of view. A software manager who wants to use a prediction model would desire to use an accurate one.

### **12. Transparency**

Transparency of a prediction system reflects the visibility of the prediction process to the software engineer/expert. Interaction or collaboration between the prediction system and the end-user/expert is of great importance, especially for maintenance purposes. If a system is transparent, an expert can easily evaluate and add his own knowledge to improve performance of the model, because it would be possible to see and understand the processes involved. Empirical research has indicated that experts coupled with prediction systems out-perform either prediction systems or experts alone [37].

### 13. Extendibility

Extendibility reflects the ability of a prediction system to accommodate changes to its model, such that it will be useful for predicting effort required for other activities of software development, e.g. maintenance, testing etc. For example, a prediction system that uses an underlying model in such a way that the prediction process expects a specific type of input, might not be useful on extending it to other activities for which such inputs are not defined.

It is worth noting that this set of attributes can serve as a fundamental groundwork of what needs to be measured when assessing and comparing prediction systems. A set of metrics can be built on top of this groundwork to quantitatively assess prediction systems. This type of metric would help a software engineer in decision making as to *what* prediction system to use *when*, through quantitative measures. This concept is briefly discussed in the next section

### 3.3 Evaluation of Prediction Systems – Sample Rating Scheme

As a demonstration of the concept, we present here a sample rating-scheme (Table 1) that could be used in developing a metric for choosing prediction systems. The scaling used is not just an ordinal scale measure since the same

uniform values (e.g. 0-3) are used across all the attributes set, summing over the ratings does not violate Fenton and Pfleeger [16] definition of admissible transformation<sup>1</sup>.

**Table 1: Rating scheme for soft computing based prediction systems**

	<b>Attributes (<math>w_i</math>)</b>	<b>Very Low (0)</b>	<b>Low (1)</b>	<b>Nominal (2)</b>	<b>High (3)</b>
1.	Underlying Model:	Totally rely on the underlying model	Heavily rely on the underlying model	Moderately rely on the underlying model	No reliance on the underlying model
2.	Trainability				
3.	Adaptability:				
4.	Sensitivity:	Totally insensitive	Moderately sensitive	Heavily sensitive	Totally sensitive
5.	Aspect Coverage:		...		
6.	Spectrum Coverage:				
7.	Implementation Model:		...		
8.	Input Data:				
9.	Knowledge Acquisition and Data Source:	⋮	...		⋮
10.	Complexity of the Model:				
11.	Accuracy:				
12.	Transparency:	Not transparent	Moderately transparent	Heavily transparent	Totally transparent
13.	Extendibility:		...		

---

<sup>1</sup> In Chapter 3 of the text, Fenton and Pfleeger stated that addition and subtraction are not meaningful on ordinal scale measures. An example in page 261 of the book reveals that the attributes used in the example are not uniformly rated. Since the rating scheme used here is uniform on all attributes, it suggests an *interval scale* measure and our addition would be adjudged meaningful.

A metric to assign quantitative value for the assessment of a prediction system is given in Equation 2. The result aggregates weights derived from assessing a particular prediction system. In order to aid the decision making process better, we present an evaluation scheme that takes the importance of an attribute to a particular prediction problem at hand into consideration.

The importance of an attribute to a particular prediction problem is context-dependent. A software engineer, while having the current problem in mind can assign a value between 0 and 5, for instance, to each criterion according to its importance (Table 2).

**Table 2: Ranking of evaluation criteria according to importance to a project**

	Attributes	Importance ( $\delta_i$ )
1.	Underlying Model:	0
2.	Trainability	
3.	Adaptability:	2
4.	Sensitivity:	
5.	Aspect Coverage:	:
6.	Spectrum Coverage:	
7.	Implementation Model:	0
8.	Input Data:	
9.	Knowledge Acquisition and Data Source:	5
10.	Complexity of the Model:	:
11.	Accuracy:	5
12.	Transparency:	
13.	Extendibility:	:

$$R = \sum_{i=1}^n \delta_i \cdot w_i$$

**Equation 2**



Any value derived for the rating,  $R$ , can serve as a basis for decision making on which prediction system to use. Thus, depending on the definition of each entry in the rating scheme, and the weighting scheme employed for the importance of the attributes, a high score,  $R$ , for a prediction system might signify the suitability of the prediction system to the prediction problem at hand.

### 3.4 Related Work on Prediction Systems Attributes

Boehm [3] presented a set of useful criteria (attributes) for evaluating the utility of software cost models. These criteria were briefly described [3] and the COCOMO-81 model was discussed [3] in light of the criteria. The attributes are targeted at model-based estimation methods. Although, soft computing based techniques are model-free, yet we find correlation between the definitions of some attributes identified, and those in our set.

In comparing three machine-learning techniques (Neural Networks (NN), Case-Based Reasoning (CBR), and Rule Induction (RI)) for effort prediction, Mair *et al.* [37] considered three factors: *accuracy*, *explanatory value* and *configurability*. These factors are subsumed in our attributes set, with their *explanatory value* synonymous to our *transparency*, *configurability* same as *complexity*, while *accuracy* remains as defined in our proposed set.

While discussing qualitative measures of prediction systems, Burgess and Lefley [9] observed that apart from accuracy, other evaluation factors for a prediction system that could be important to non-expert practitioners are necessary. They identified some of these factors [9].

All the evaluation factors by Burgess and Lefley are subsumed in our attributes set, with the exception of *likelihood of convergence*. Burgess and Lefley did not give detailed explanations for this factor (i.e. likelihood of convergence), and we cannot ascertain if any of our attributes captures it.

Briand *et al.* [8] provided an evaluation ground to compare and assess resource estimation methods. The study identified three categories of evaluation criteria, with fifteen attribute sets. The first category deals with criteria for assessing models and their estimates (Model and Estimate Criteria). The second deals with criteria for assessing the process to build the final estimate (Estimation Method Criteria), while the third category deals with criteria for assessing the applications of an estimation method to obtain resource estimate (Application Criteria). In a comparative evaluation of estimation methods, some of the criteria are not so general to be applicable in all circumstances. However, soft computing based techniques were not discussed in their attributes set. Most of their attributes are subsumed in ours except for *calibration*, *repeatability* and *assumptions*. Calibration, for

example is specific to generic model-based methods and not applicable to soft computing based techniques, while the other two are not considered important for our purpose. Our proposed rating scheme offers a much more promising decision making support than the binary (*low - high*) rating scheme discussed in the work of Briand *et al.*

### **3.5 Critical Survey of Existing Soft Computing Based Approaches to Effort Prediction**

In this section, we present a summary discussion of some existing works based on our set of attributes. Further details can be found in [59]. The list of considered approaches in our study is not exhaustive, but we gave attention to those works we considered significant and more recent as regards the subject of discussion. According to our definition of trainability, all the approaches considered are trainable. Therefore, we would not consider “trainability” during the discussion of the different approaches, since the attribute is not a discriminator as regards the considered works.

#### **3.5.1 Fuzzy Logic Based Approaches**

##### **3.5.1.1 Musilek *et al.* [41]**

The approach here looked at fuzzification of the basic COCOMO model at two different levels of detail. The first level called simple f-COCOMO is concerned about representing *size* of software project as fuzzy set while the

coefficients representing *mode* remain crisp values. The second level, called the augmented f-COCOMO provides a representation of the modes of software development as singleton fuzzy sets. Evaluation of the approach based on our attributes is given in Table 3.

**Table 3: Evaluation of Musilek *et al.* [41] approach to effort prediction**

	Attributes	Approach
1.	Underlying Model:	The underlying model in the proposed work is an algorithmic cost model; the basic COCOMO model.
2.	Adaptability:	The approach is <i>not</i> adaptive.
3.	Sensitivity:	The <i>size</i> representations for the two levels are based on fuzzy sets. The approach is able to reduce the sensitivity to imprecision in the input data. Although, the discrete number representation of <i>mode</i> does not take care of imprecision.
4.	Aspect Coverage:	It simply covers <i>size</i> and <i>mode</i> as defined by the underlying model, so no attempt to address other aspects of the development process, such as technology and reuse.
5.	Spectrum Coverage:	It covers a wide spectrum of systems since it takes <i>mode</i> of software development into consideration.
6.	Implementation Model:	<p>The implementation approach is not in anyway better than using the COCOMO model equation itself. Because it is natural for anyone to specify a range for the <i>size</i> and use the model to calculate a range for effort based on the input range. A typical fuzzy system should model the relationship between the input/output variables with, for example, using fuzzy rules rather than the model equation for calculation.</p> <p>The fuzzy set used to represent <i>mode</i> of software development in their augmented f-COCOMO are singletons, which reduces to using crisp values for linguistic variables.</p> <p>We are of the opinion that if the approach is modified to use non-singleton fuzzy sets, uncertainties in classifying projects into different <i>modes</i> will be better modeled and the result should be more promising. We have addressed this in our framework to be presented in Chapter 4 of this work.</p>
7.	Input Data:	The input required is <i>size</i> measured in KLOC and <i>mode</i> . The two input values are not so difficult to obtain. First, the features for classifying systems into modes are already defined in [3], and rough estimates of size can be done early in the development process.
8.	Knowledge Acquisition and Data Source:	Knowledge acquisition is through manual procedures, because experts are required to give accurate degree of membership to each <i>mode</i> in the augmented f-COCOMO. This precision required for membership degree does not offer enough flexibility typical of fuzzy systems, as it can be difficult giving such values.
9.	Complexity of the Model:	The idea behind the model is simple but not same for an end-user who needs knowledge of fuzzy logic to be able to understand and apply the idea proposed.
10.	Accuracy:	It is not straightforward to judge the accuracy of the model.
11.	Transparency:	The approach is to some extent transparent. Nevertheless, a domain expert/end-user cannot offer any enhancement.
12.	Extendibility:	The model is not extendible since it is strictly dependent on the existence of the underlying COCOMO model, and the input to the prediction system must be <i>LOC</i> and <i>mode</i> .

### **3.5.1.2 Idri and Abran [23]**

The cost drivers (CD) of immediate COCOMO'81 model were fuzzified to take care of the very sharp transition between two different intervals defined for a single CD. Fuzzy sets were defined to model the different categories, such that two analogous projects, in different categories, would not have a very large difference in their effort estimates. The approach was more of a sensitivity analysis on the cost drivers. Details of their experiments on each of the studied cost drivers were not reported, except for the DATA cost driver. However, four of the 15 cost drivers used in intermediate COCOMO, namely: RELY, CPLX, MODP and TOOL were not studied in their proposed approach. The reason given was that the relative description given for categorizing projects based on these four attributes are not sufficient to aid fuzzy sets definition. We discuss their approach based on our proposed attributes in Table 4.

**Table 4: Evaluation of Idri and Abran [23] approach to effort prediction**

	Attributes	Approach
1.	Underlying Model:	The underlying model in the work is the intermediate COCOMO'81 cost model.
2.	Adaptability:	The approach described is <i>not</i> adaptive.
3.	Sensitivity:	The approach is very sensitive to changes in input data since fuzzification is only applied to the cost drivers and not to the other inputs.
4.	Aspect Coverage:	There is low aspect coverage as only cost drivers were considered.
5.	Spectrum Coverage:	The approach recognizes the different classes of projects, e.g. the <i>organic</i> , proposed by Boehm [3], but is not useful for system that does not use cost drivers. It therefore exhibits low spectrum coverage.
6.	Implementation Model:	Artificial datasets were defined to evaluate the approach. The actual cost estimate is assumed to come from the original model equation. The experiment was more of a sensitivity analysis.
7.	Input Data:	The type of input data required are categorical attributes (e.g. high, very high, etc) that would be amenable to fuzzy sets definition. These data are not easy to get, since they are available much later in the project when such information becomes available.
8.	Knowledge Acquisition and Data Source:	There is no knowledge acquisition in the proposed approach. Fuzzy rules formulation are not involved and no knowledge base development.
9.	Complexity of the Model:	Evaluating the complexity is not straightforward since it is not a complete effort prediction system.
10.	Accuracy:	The approach discussed is not such an effective approach as it concentrates only on the adjustment factors, while it ignores the major factors i.e., <i>size</i> and <i>mode</i> of software as defined in [3]. Imprecise input data (size and mode) to the COCOMO model at the top level nullifies the whole essence of the approach.
11.	Transparency:	There is no complete transparency in their approach. No modalities for defining fuzzy sets, and only one single example (i.e. DATA) cost driver was consistently given, which intuitively lends itself to trapezoidal fuzzy set definition.
12.	Extendibility:	The approach is not extendible since it is not useful without the underlying model. Again, it does not cover all classes of software project. Thus, extendibility is <i>very low</i> .

### 3.5.1.3 MacDonell *et al.* [34][35][36]

A tool, FULSOME was developed using fuzzy logic to help software metricians in data acquisition, model expression and knowledge gathering issues. It provides interfaces for data entry, membership function construction, rule creation and output of the inference process, in similar fashion to the MATLAB Fuzzy Logic Toolbox [17]. The experts can

perform the membership function definition and rules formulation manually, as it is the case in the MATLAB Toolbox.

A module that is not part of the FULSOME tool is provided as an alternative to membership function generation and rule formulation. This module can extract membership functions and rules automatically from a given set of data, when supplied with the membership function type and the number of membership functions desired. These rules are not adaptive, and are said to be useful only as a first draft for the expert to extend. In their words, *“automatically generated fuzzy systems are rather worse than the regression result.....due to insufficient rule coverage.....would be a useful first draft for an expert to extend”* [36]. It is not clear from their discussion whether the problem lies with fuzzy systems (very unlikely) or their approach. A more matured experiment is needed to establish this fact. The rules generated are also not sensitive to user needs. When a user supplies rules in a bid to extend the initial rules generated, it will destroy their clustering method because the added rules were not used in the initial clustering to extract membership functions. A summary evaluation of the approach is given in Table 5.



**Table 5: Evaluation of MacDonnel *et al.* [35] approach to effort prediction**

	<b>Attributes</b>	<b>Approach</b>
1.	Underlying Model:	It is built on expert judgment
2.	Adaptability:	The tool implemented is not adaptive.
3.	Sensitivity:	Since it is a straightforward implementation of fuzzy inference system, it should be able to adequately model imprecision in input, which translates to low sensitivity.
4.	Aspect Coverage:	It can have wide aspect coverage depending on what features/attributes of the project that the domain expert adjudged to be important.
5.	Spectrum Coverage:	Spectrum coverage is left to the discretion of the domain expert as discussed for aspect coverage.
6.	Implementation Model:	The implementation model is a fuzzy inference engine in the form of a tool.
7.	Input Data:	Using the system requires historical data or the availability of experts to supply the data, based on experience.
8.	Knowledge Acquisition and Data Source:	Updating the knowledge base in the system is manually done by a domain expert, as it simply provides an interface for the user to update the system with rules. The sources of data are through experts and historical data database.
9.	Complexity of the Model:	The system is not so complex since it follows the normal fuzzy inference procedures.
10.	Accuracy:	Depends on accuracy with which the expert can determine the relationships between project features and assign relationships using fuzzy variables.
11.	Transparency:	The approach taken is transparent because it allows software managers to have the flexibility to identify and define features believed to affect software projects.
12.	Extendibility:	It is extendible; the only changes required are the features of the new target domain.

### 3.5.2 Neural Networks Based Approaches

#### 3.5.2.1 Wittig *et al.* [74]

Wittig *et al.* examined the performance of back-propagation artificial neural networks in estimating software development effort. Two different experiments were carried out; first on simulated data generated from a metric model (SPQR/20<sup>2</sup>) and the second set of experiments involve actual

---

<sup>2</sup> SPQR/20 is an estimation tool for function point counting techniques. SPQR/20 addresses two criticisms of Function Point Analysis. For further details on SPQR/20, the reader is referred to [74]

development data. In all cases, the size of software was measured in function points (FPs) and project effort was measured in development hours. For a generalized guidelines used in developing the neural networks experiments, we refer the reader to [74].

In order to generate simulated project data, sets of input values were generated using a random number generator for 1000 projects, and SPQR/20 was then used to generate the project development effort. These data were used to train the neural network to generate effort estimates from the function point size.

The second set of experiments was carried out on two different actual project data; first project data consists of data from ten Canadian organizations while the second sets of data were from Australian Software Metrics Association (ASMA) database of development project information. Table 6 gives a summary of our evaluation based on our attributes:

**Table 6: Evaluation of Wittig *et al.* [74] approach to effort prediction**

	<b>Attributes</b>	<b>Approach</b>
1.	Underlying Model:	The underlying model is Function Points (FPs) model for size.
2.	Adaptability:	It is not adaptive.
3.	Sensitivity:	No defined/detailed approach to evaluate sensitivity, a neural network models the problem at hand. It is expected that their approach will be highly sensitive to the input data.
4.	Aspect Coverage:	The input to the model is <i>size</i> of the software project; the technique does not have wide aspect coverage.
5.	Spectrum Coverage:	The prediction system did not cover different types of systems.
6.	Implementation Model:	Implementation was based on back-propagation training algorithm
7.	Input Data:	The prediction system takes the size in FPs form as input.
8.	Knowledge Acquisition and Data Source:	Knowledge acquisition is automatic through training. The data sources are historical databases and automatically generated data. The approach is heavily reliant on data.
9.	Complexity of the Model:	The approach is overly complex, typical of neural networks. The authors had to carry out another systematic approach to determine topology, parameter settings etc., in order to perform a second experiment with adjustment factors on ASMA data.
10.	Accuracy:	The technique was able to model the systems considered with a desirable accuracy level. In general, the performance of the model is subject to the quality of data on which it is trained.
11.	Transparency:	The prediction system is not transparent
12.	Extendibility:	Since the approach is dependent on the current project data, extending the system is about building another prediction system.

### 3.5.2.2 Other Selected Neural Networks Based Approaches [7] [6] [60]

Samson *et al.*[60] applied a neural network model, Cerebellar Model Arithmetic Computer (CMAC) to prediction of effort from software code size (KDSI). CMAC is a perceptron and function approximator developed by Albus. This neural network was trained on Boehm's COCOMO data set in order to predict effort from size, in the same manner regression techniques are used to fit a line to a data set for prediction purposes. The results of the prediction performed better than linear regression on the data set.

Boetticher [6] conducted over 33,000 different experiments using neural networks on empirical data collected from separate corporate domains. The experiments assessed the contribution of different internal product metrics (size, vocabulary, complexity, and object) to programming effort, using neural networks.

Boetticher [7] in a later work to the previous described, adopted a bottom-up approach to conduct series of neural networks experiments on data from two separate domains. A bottom-up approach uses data from products rather than projects. A size-based metric, Source Lines of Code (SLOC) extracted from an actual program was used as the only input for the neural network to predict project effort. The approach is appropriate for methodologies creating prototypes very early in the life cycle. Estimate of SLOC for each component serves as input to a trained neural network, and calculated results could be summated to generate a *project programming effort (PPE)*. The author reported that PPE can replace SIZE parameter in COCOMO II cost model, although we do not see the advantage of this replacement, since PPE is already a calculated *effort* from size in SLOC.

### 3.5.3 Evolutionary Computation Based Approaches

#### 3.5.3.1 Burgess *et al.* [9]

Burgess *et al.* evaluated the potential benefits of applying genetic programming (GP) to software effort estimation when compared with previously published approaches such as artificial neural networks and case based reasoning, in terms of accuracy, explanatory value and ease of configuration/use. Their investigation was based on one of the data set used by Wittig *et al.* [74] from some Canadian software organizations. The data set comprised 10 features: nine independent and one dependent (effort) [9]. The independent features refer to those factors that are known to influence software development effort, which is the dependent variable. Some of these independent features include function points, development environment, team experience, etc.

The research is an extension of the work on the same approach by Dolado [14]. Software effort estimation was formulated as a symbolic regression problem. The authors merely reported that the results obtained depend on the fitness function used, but no information regarding such fitness functions.

In carrying out their comparison based on accuracy, GP system were said to be consistently more accurate but does not converge as consistently as the ANN to a good solution. In terms of transparency, GP produced a more transparent solution as it uses algebraic expression. Finally, in terms of ease

of configuration, ANN are always known to be very complex, however, GP has many parameters as well, such as choice of functions, crossover and reproduction rates, setting maximum trees sizes and depths, etc. Discussion of the approach based on our attributes is presented in Table 7.

**Table 7: Evaluation of Burgess *et al.* [9] approach to effort prediction**

	<b>Attributes</b>	<b>Approach</b>
1.	Underlying Model:	The prediction system is not based on any existing estimation model.
2.	Adaptability:	It is not adaptive.
3.	Sensitivity:	Evaluating sensitivity is not clear in their discussion. It is however likely to suffer from the same problems as COCOMO and other regression-based approaches.
4.	Aspect Coverage:	Aspect coverage depends on the underlying data set. The data set on which the experiments were carried out is function points based.
5.	Spectrum Coverage:	There was no special attempt at identifying different classes of systems.
6.	Implementation Model:	Implementation was based on Genetic Algorithms concepts and written in C.
7.	Input Data:	The inputs to the model are function points.
8.	Knowledge Acquisition and Data Source:	The knowledge acquisition is automatic and data source is from historical data sets.
9.	Complexity of the Model:	The system is very complex. So much effort is required in setting it up and training.
10.	Accuracy:	They reported some fairly accurate results.
11.	Transparency:	It offers some degree of transparency, but not to the level that accommodates user's interaction.
12.	Extendibility:	Randomly constructed trees were said to be dependent on the type of problem being solved. Their prediction system is therefore not extendible, but the approach is.

### 3.5.4 Neuro-Fuzzy Logic Based Approaches

#### 3.5.4.1 Hodgkinson *et al.* [20]

The authors built an adaptive neuro-fuzzy system by combining learning and modeling aspects of neural networks with the linguistic properties of fuzzy

systems. Neuro-fuzzy models produced fuzzy sets and rules for each training dataset.

They have two input variables into the neural network; *size* in thousands lines of code (KLOC) and *duration* (months), while the output is the *effort* (man-months) expended during the project. Each of the two input variables and the output variable has number of nodes that depends on the classification of the training dataset, which the network evolves, e.g., *small*, *medium* and *large* sets. The network is trained in a similar fashion to that of a feed forward ANN.

The rules evolved using the network clearly indicates that we still need more visibility and explanations for the rules. For example, in the same domain as reported in their work, the dataset contains a classification involving two categories that are not different classes according to the rules (e.g. software projects of sizes *low* and *medium* that require same duration would give same effort estimate). This implies that some ranges of size of software project are not really taken as major predictor of effort. For example, we have the following as part of the rules evolved:

*If size is **low** AND duration is low THEN effort is low (0.35) OR effort is high (0.65)*

*If size is **medium** AND duration is low THEN effort is low (0.35) OR effort is high (0.65)*

where (0.35) and (0.65) represent the confidence levels of the rule with respect to the two consequent fuzzy sets.

In our own view, two different categories of sizes should not result in the same required effort, and if at all, at least not with the same confidence level attached to the rules as it is obtained in the two rules above.

Ordinarily, we also believe that modularity of knowledge is desirable in software cost estimation where the duration could be derived from the estimated effort, since the nature of a software project can also contribute to duration.

An evaluation of this approach to effort prediction is given in Table 8



**Table 8: Evaluation of Hodkinson *et al.* [20] approach to effort prediction**

	<b>Attributes</b>	<b>Approach</b>
1.	Underlying Model:	The approach did not use any of the existing prediction models.
2.	Adaptability:	The system is not adaptive.
3.	Sensitivity:	As a fuzzy based approach, it should be less sensitive to the input data. Fuzzy sets are evolved to model the training dataset.
4.	Aspect Coverage:	It did not take the development process and environment into consideration, which implies that it does not possess wide aspect coverage.
5.	Spectrum Coverage:	The approach completely ignores the role of system complexity in their effort estimation, unlike <i>mode</i> as defined in COCOMO. The classification of training dataset as evolved by the neural network algorithm is just a reflection of the training data distribution and not based on the project/system complexity.
6.	Implementation Model:	The implementation model employed is a hybrid neuro-fuzzy model that combines the learning aspect of neural network with linguistic properties of fuzzy systems. Two-phase approach was used: Training phase and Estimation Phase.
7.	Input Data:	Both size and duration are taken as input. Getting the input data can be difficult, because estimating duration is surrounded with uncertainty level close to effort estimation.
8.	Knowledge Acquisition and Data Source:	Knowledge acquisition in the system is automatic during training. The source of knowledge is through historical dataset.
9.	Complexity of the Model:	The model is a complex model. It requires the system to be represented as a feed-forward network to be amenable to neural network training. Thus, determining the topology of the network and other related issues are not easy.
10.	Accuracy:	Evaluating accuracy of the model requires further experiments.
11.	Transparency:	The model is not transparent to the end-user. The rules in the system are evolved based on data classification, all carried out by the network.
12.	Extendibility:	The system would only work on the data on which it was trained; extending it to carry out prediction not based on the trained data is tantamount to starting from scratch.

### 3.5.5 Neuro-Genetic Based Approaches

#### 3.5.5.1 Shukla, K. K. [64]

The author presented a genetically trained neural network for effort prediction based on historical data (COCOMO and Kemerer datasets). The approach uses a multi-layered feed-forward neural network with 39 input neurons and one output neuron representing the predicted effort in person-

months. The 39 input neurons represent 39 project attributes (features) identified by Boehm [3].

The problem requires a search for an optimum weight vector that globally minimizes the error function (a performance metric) over a training set.

GA was utilized in solving the global optimization problem to train the neural net to predict development effort. The application of GA to the NN training required binary representation of the weight parameters.

Simulation experiments were conducted with the feed-forward network using back-propagation (BP), quick-propagation (QP) and genetically trained NN (GANN) to obtain the relative performance. GANN was reported to have clearly out-performed the two algorithms in all folds: convergence, accuracy, generalization ability and sensitivity to weight initializations.

An evaluation of this approach to effort prediction is given in Figure 9.

**Table 9: Evaluation of Shukla, K. K. [64] approach to effort prediction**

	<b>Attributes</b>	<b>Approach</b>
1.	Underlying Model:	The prediction system is not based on an existing estimation model.
2.	Adaptability:	The system is not adaptive.
3.	Sensitivity:	Sensitivity of the prediction system to input data was not addressed, and since precise input are required for each of the project features, it is expected that it will be highly sensitive to input.
4.	Aspect Coverage:	The coverage of the prediction system depends on the coverage of the underlying data set.
5.	Spectrum Coverage:	There was no special attempt to identify different classes of systems. Although, the 39 features considered might have such coverage.
6.	Implementation Model:	The simulator was developed using MS Visual C++, with GA to train the Neural Network.
7.	Input Data:	The approach does not completely describe a prediction system, as it did not explicitly identify the input required to perform predictions.
8.	Knowledge Acquisition and Data Source:	The simulator implements automatic knowledge acquisition using the underlying historical datasets. Knowledge is acquired through weight adjustments.
9.	Complexity of the Model:	The system is complex since it requires expertise in setting up the network and determining representation scheme for the chromosomes.
10.	Accuracy:	The author reported better accuracy as compared to other NN training approaches.
11.	Transparency:	It is not transparent
12.	Extendibility:	It is extendible. Identification of the attributes describing an activity is all that is required.

### **3.6 Conclusion on Existing Soft Computing Based Prediction Approaches**

Our critical survey of state-of-the-art in effort prediction with soft computing based techniques, using our attributes set, reveals some shortcomings related to existing soft computing based effort prediction approaches. These problems include, but are not limited to the following:

1. Most of the approaches are formulated based on historical data, which are hardly available. Expert knowledge is a good alternative that some of the approaches have incorporated.
2. All investigated approaches lack adaptability. Therefore, they cannot cope with the continuous change in development technologies and environments.
3. Most of the approaches are not transparent enough to accommodate expert's knowledge in a well-defined manner. They are also adaptive only during training.
4. There are immature experiments in soft computing based approaches to effort prediction.
5. The various attempts made to address the fuzziness of the COCOMO model did not address the integration of the different

aspects of the COCOMO into a single framework. No rules formulation procedures were discussed to model the relationships between predictor variables (i.e. size, mode and cost drivers) and effort. Fuzzification of the COCOMO model only looked at either basic part of the model or the cost drivers in isolation.

6. There does not exist a framework that incorporates both adaptability and adequate transparency.

Properly addressing these problems would position soft computing based prediction techniques as models of choice for effort prediction, considering the promising features already present in them.

Our attempt at addressing some of the issues lead to the fuzzy logic based framework that we present in the next chapter.

## ***CHAPTER 4***

### **THE RESEARCH APPROACH AND FRAMEWORK**

#### **4.1 Introduction**

This chapter provides details of our framework, a reference architecture that will guide us through this research. The framework follows from our attempt to address some of the problems resulting from our critical survey in Chapter 3.

#### **4.2 The Problem**

Most significant of the problems identified in concluding our critical survey of chapter 3, is the lack of adaptive soft computing based effort prediction systems that provide complete transparency to the prediction system building strategies, such that experts could easily augment with their knowledge while building and using the prediction system.

Some of the approaches like the neuro-fuzzy technique by Hodgkinson *et al.* [20] are also not transparent enough to allow validation and accommodate expert opinion. Validation is essential to users that will want to know the

origin of figures estimated, while expert opinions are necessary when tuning a prediction system to a new environment.

In addition, efforts at handling the imprecision problem surrounding one of the most widely used algorithmic model, COCOMO, has not been appropriate, since the model was not addressed as a whole. Thus, integrating the individual component of the model into a single prediction system remains an open question.

In order to address these problems, we have developed a fuzzy logic based framework that is built upon an existing cost estimation model. This framework to be introduced later in this chapter addresses these problems by:

- Providing a framework that is well-defined and can accommodate expert knowledge
- Proposing and implementing a well-defined procedure to fuzzify and integrate the various components of the COCOMO model, and
- Implementing training algorithms to incorporate adaptability

### **4.3 Research Methodology**

This section presents the intermediate COCOMO model used in this research, our approach to solving the problems discussed in the previous section, and the framework for effort prediction.

#### 4.3.1 The Intermediate COCOMO Model

The Intermediate COCOMO model as stated in Chapter 1 is the most widely used version. Intermediate COCOMO model, has estimation accuracy that is substantially greater than the basic version, and comparable to what obtains with the detailed version that incorporates additional cost drivers [23]. It takes as input<sup>3</sup> the estimated size of the software product in thousands of Delivered Source Instructions (KDSI) adjusted for code reuse [3], [11], the project development mode given as a constant value  $B$  (also called the scaling factor) that depends on one of the three categories of software development modes (organic, semi-detached, embedded), and 15 cost drivers. The development mode can take only three values,  $\{1.05, 1.12, 1.20\}$ , which reflect the difficulty of the development. The estimate is adjusted by factors, called *cost drivers* that influence the effort to produce the software product. Cost drivers have up to six levels of rating: Very Low, Low, Nominal, High, Very High, and Extra High. Each rating has a corresponding real number (effort multiplier), based upon the factor and the degree to which the factor can influence productivity.

The estimated effort in Person Months for the intermediate COCOMO is given as:

---

<sup>3</sup> The Basic and Detailed versions take *size* and *mode* as input as well. In addition, the level of details required for cost driver's definitions is higher for Detailed COCOMO than Intermediate.



$$\text{Effort} = A \times [\text{Size}]^B \times \prod_{i=1}^{15} EM_i \quad \text{Equation 3}$$

The constants and scale factors for the different modes are given in Table 10 and the cost drivers are given in Table 11.

**Table 10: COCOMO mode coefficients and scale factors values**

<i>Mode</i>	<b>A</b>	<b>B</b>
Organic	3.2	1.05
Semidetached	3.0	1.12
Embedded	2.8	1.2

**Table 11: COCOMO Cost Drivers**

Category	Cost Drivers	Symbol	i
Product			
	Required software reliability	RELY	1
	Data base size	DATA	2
	Product complexity	CPLX	3
Platform			
	Execution time constraints	TIME	4
	Main storage constraint	STOR	5
	Virtual machine volatility	VIRT	6
	Computer turnaround time	TURN	7
Personnel			
	Analyst Capability	ACAP	8
	Applications experience	AEXP	9
	Programmer capability	PCAP	10
	Virtual machine experience	VEXP	11
	Programming language experience	LEXP	12
Project			
	Use of modern programming practices	MODP	13
	Use of software tools	TOOL	14
	Required development schedule	SCED	15

The effort multipliers corresponding to the cost drivers are incorporated into the effort estimation formula by multiplying them together. The numerical value of the  $i^{\text{th}}$  cost driver is  $EM_i$  and the product of all the multipliers is called the estimated adjustment factor (EAF). The actual effort in person-

month (PM),  $PM_{total}$  is the product of the nominal effort times the EAF, as given below:

$$PM_{total} = PM_{nominal} \times \prod_{i=1}^{15} EM_i \quad \text{Equation 4}$$

Detailed definitions and rating scales of the intermediate COCOMO cost drivers can be found in [3]. Our concentration in this research work is on the intermediate version, for reasons already discussed.

Many traditional models have been said to perform poorly when it comes to cost estimation, COCOMO'81 is said to be most plausible [41], best known [29], and most cited [20] of all traditional models. Over a dozen commercial COCOMO '81 implementations are available [13]. The original COCOMO database is also readily available for validation.

A new version of the COCOMO model tailored to current software life-cycle processes (COCOMO II) has been formulated, with some calibration done and still in progress [13].

#### 4.3.2 Fuzzy Logic Approach to COCOMO

Vagueness is present in all parameters entering the COCOMO model. The exact size of software project to be developed, for instance, is difficult to

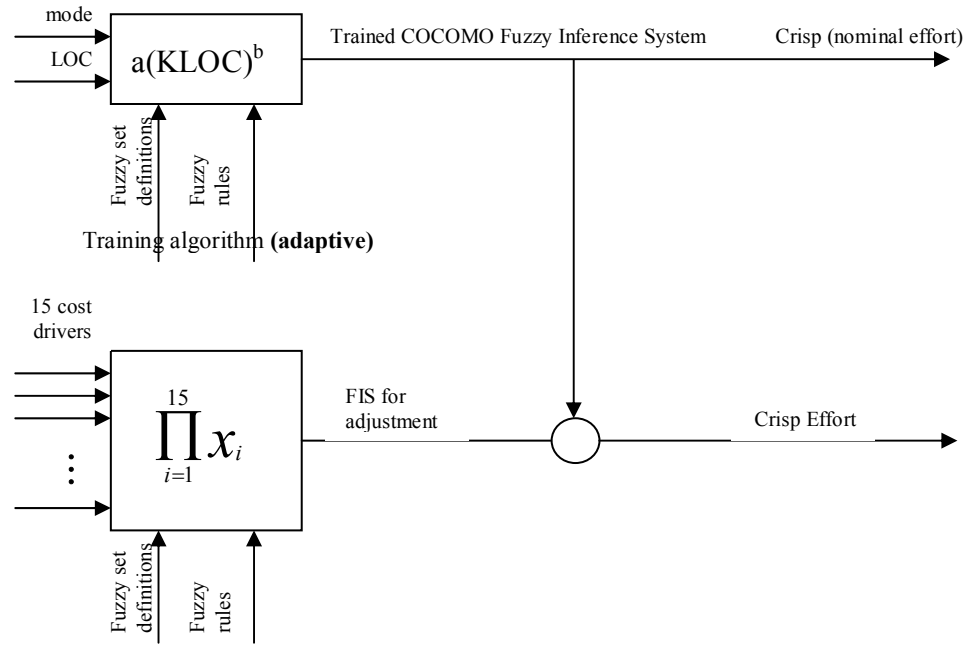
estimate to precision at an early stage of the development process when correctness and precision of the size estimate are limited. Opportunities are also not given to consider software projects that do not entirely fall into exactly one of the three modes of development. In addition, the cost drivers are categorical (e.g. high, low etc.), but determining the category that applies to a project for most cost drivers depends on crisp values estimates that might not entirely fall under a single categorical class. A clearer explanation on these categorical values appears while discussing cost drivers in our framework in the next sub-section. In all these input parameters, fuzzy sets can be employed to handle the imprecision.

Various attempts have been made to address the fuzzy nature of some aspects of the COCOMO model, with prominent ones including Idri *et al.* [23], Musilek *et al.* [41], and Pedrycz *et al.* [51]. Discussion of these works was done by Saliu and Ahmed [59] and Chapter 3 of this thesis.

As stated in the concluding part of Chapter 3, these works have not looked into the integration of the individual component of the COCOMO model. Our framework to be presented in the next section addresses this problem.

### 4.3.3 The Proposed Framework

Our adaptive framework uses intermediate COCOMO as the base cost model to solve the major problem of data scarcity and fuzzy rules training. The framework is shown in Figure 8.



**Figure 8: Adaptive Fuzzy Logic Based Framework using COCOMO**

The framework identifies the two components of the COCOMO model as given in Equation 4: the nominal effort as calculated using the basic part of the model, and the effort adjustment using the effort multipliers. The two component parts are integrated in a single framework that allows fuzzy rules generation, rules training, and expert knowledge incorporation.

We desire to develop and train fuzzy rules using the model equation (or expert knowledge) for the basic part at the top to estimate nominal effort, and fuzzify the cost drivers represented at the bottom part of Figure 8. Such independent training can only be achieved only if the effect of one multiplier is independent of the effect of others. In fact, this property already exists in the model and supported by Pfleeger [52] that the computation of the effort adjustment factors is valid only when the individual adjustment factors are independent. To show that, we take the logarithms of both sides of Equation 3, to have:

$$\ln(\text{Effort}) = \ln\left(A \times [\text{Size}]^B\right) + \ln\left(\prod_{i=1}^{15} EM_i\right)$$

$$\ln(\text{Effort}) = E1 + E2$$

Where,

$$E1 = \ln\left(A \times [\text{Size}]^B\right)$$

$$E2 = \ln\left(\prod_{i=1}^{15} EM_i\right)$$

**Equation 5**

$$E1 = \ln(A) + B \cdot \ln(\text{Size})$$

**Equation 6**

$$E2 = \ln(EM_1) + \ln(EM_2) + \dots + \ln(EM_{15})$$

**Equation 7**

From Equation 6 and Equation 7, the relationships between effort and each of the predictor variables have been shown to be independent. Thus, we can

fuzzify and train the nominal part of the model equation independent of the cost drivers. Effects of the drivers on efforts are independent and can be aggregated with the predicted nominal effort. The fuzzy sets definitions take care of imprecision in input to the effort prediction model. Informal description of our approach for fuzzifying the two parts that constitute the overall effort, E1 and E2, is given below.

#### PART I: Fuzzy COCOMO for Nominal Effort

E1 reconsidered from the basic part of the COCOMO model equation, we have

$$E1 = \left( A \times [Size]^B \right)$$

Handling the imprecision in input supplied for size requires that size of software project be redefined as a fuzzy number, instead of crisp number, using suitable fuzzy sets. Using fuzzy sets, the size of a software project can be specified as a range of possible values and the closeness of each value to the actual value depends on the degree to which the value is a member of the fuzzy set describing the fuzzy number. Similarly, given the values for A and B as in

Table 10, we define fuzzy sets for each *mode* of development such that each adjacent fuzzy set overlaps with its neighbors. The overlap will exploit the power of fuzzy logic to enable us handle development projects that fall

between two modes (a situation not possible in the COCOMO model). Sample artificial datasets are generated for *size*, and the corresponding *effort* is calculated using the COCOMO model equation. For each selected size and cost calculated, membership functions are defined. We then proceed to formulate rules based on the relationship between size and effort, and train the fuzzy inference system developed using the artificial dataset.

In carrying out our experiments, we have chosen to use artificial datasets as opposed to the COCOMO database for the following reasons:

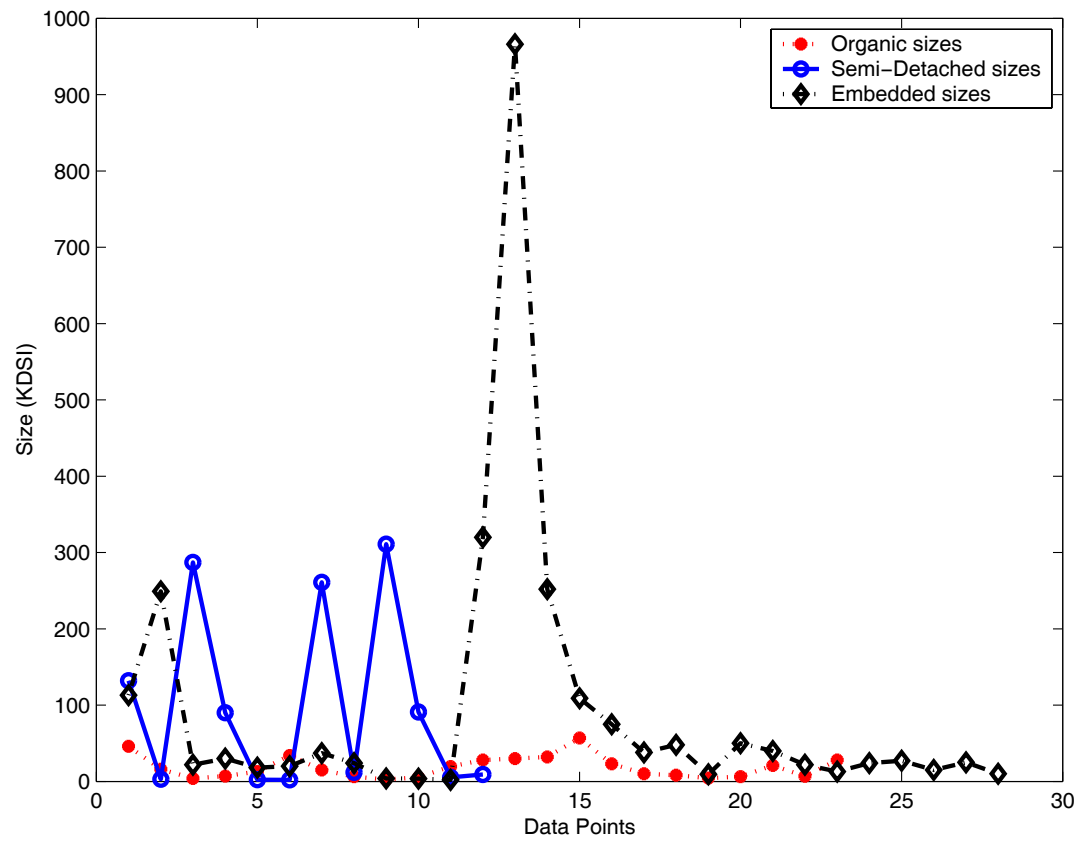
1. The COCOMO database contains 63 data points 28 Embedded, 12 semi-detached, and 23 Organic. Thus, the data is not uniformly distributed over the different modes (see Figure 9).
2. The adjusted sizes in thousands lines of code of software projects in the database are not evenly distributed. For example:
  - a. All data-points are made up of sizes below 1000KDSI, with the highest size being 966KDSI
  - b. The next highest size to the one stated above is 320KDSI (No size exists between 321 and 955)
  - c. Of the 63 data-points in the database, 53 data-points have their sizes of lines of code to be less than 100KDSI (meaning



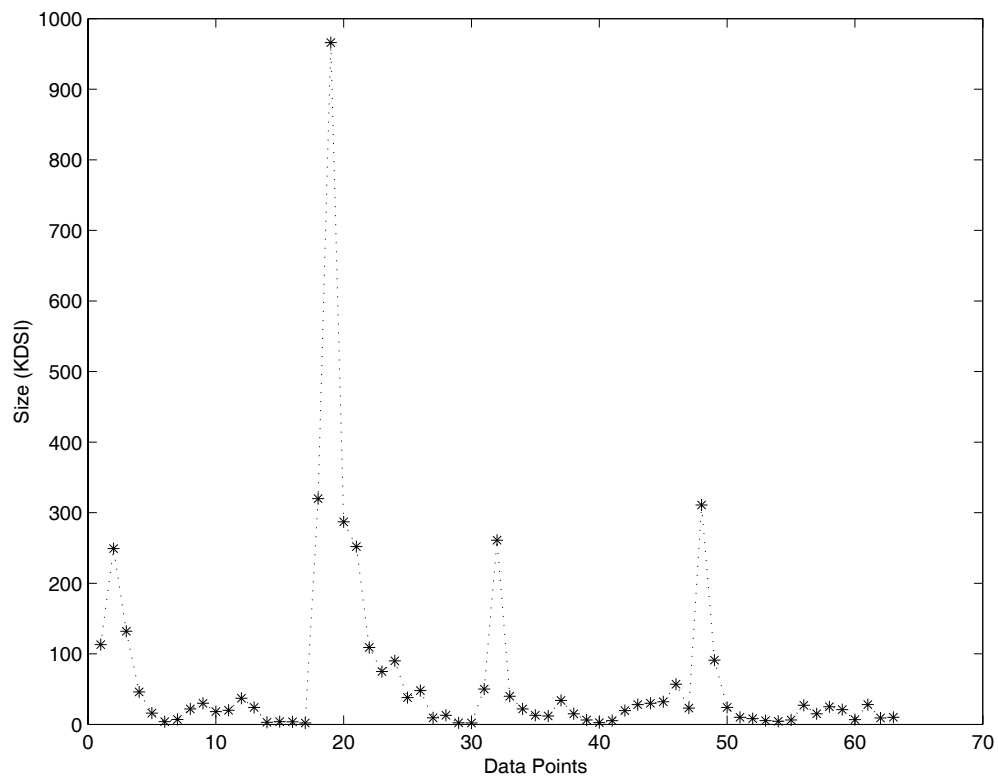
only 9 data-points have sizes between 100 and 321KDSI, and 1 data-point above 321KDSI). Please refer to Figure 10.

From the observations stated above, we will have difficulties in properly predicting effort for sizes in the range 500 -1000KDSI.

Another reason for not using the COCOMO database during training is that this way the database will be useful in validating our approach, since it is not involved in the training.



**Figure 9: Distribution of Size (KDSI) of the COCOMO Database Based on Mode of Development**

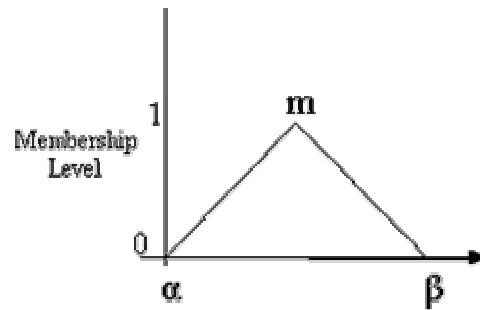


**Figure 10: Distribution of Size (KDSI) of the entire COCOMO Database of 63 Software Projects**

#### Membership Function Selection Method

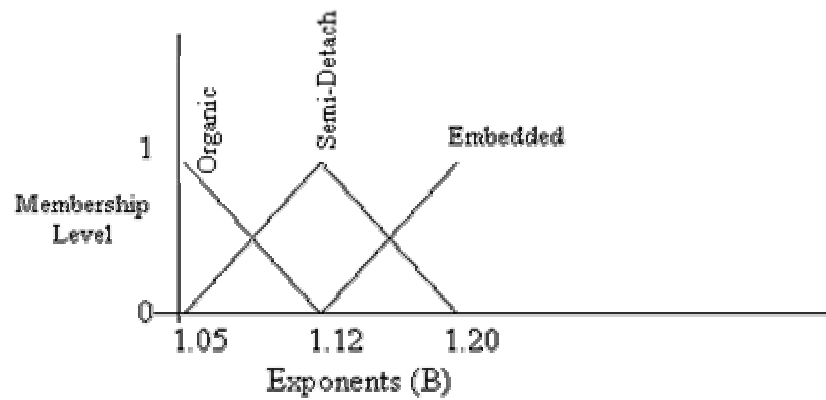
As discussed in Chapter 2, implementing a fuzzy system requires that the different categories of the different inputs be represented by fuzzy sets, which in turn, is represented by membership functions. For our problem, a natural membership function type that readily comes to mind is the triangular membership functions. It is a three-point function, defined by minimum ( $\alpha$ ), maximum ( $\beta$ ) and modal ( $m$ ) values, that is,  $\text{TMF}(\alpha, m, \beta)$ , where  $(\alpha \leq m \leq$

$\beta$ ), Please refer to Figure 11 for a sample triangular membership function, showing  $\alpha$ ,  $m$ , and  $\beta$  with center  $m$ , and support  $[\alpha, \beta]$ .

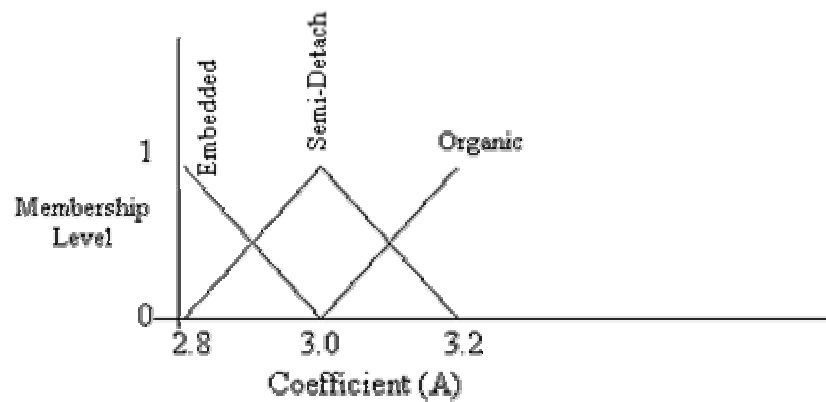


**Figure 11: A Triangular Membership Function**

The fuzzy sets definitions for the mode of development appear in Figure 12, and Figure 13 below. For example, the middle triangular fuzzy set can be used to represent the fuzzy number (*approximately* semi-detached), with center 1.12, and support [1.05, 1.20].

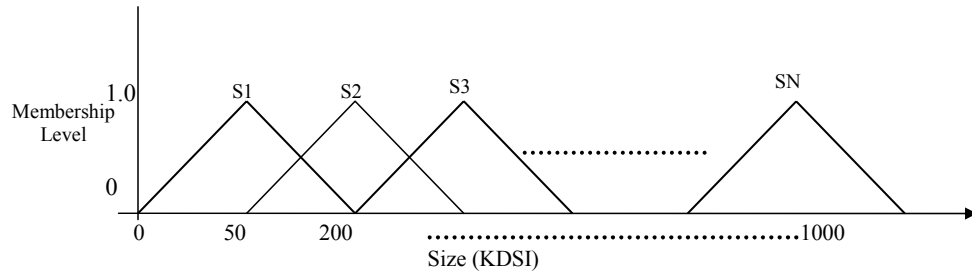


**Figure 12: COCOMO scale factors fuzzification**



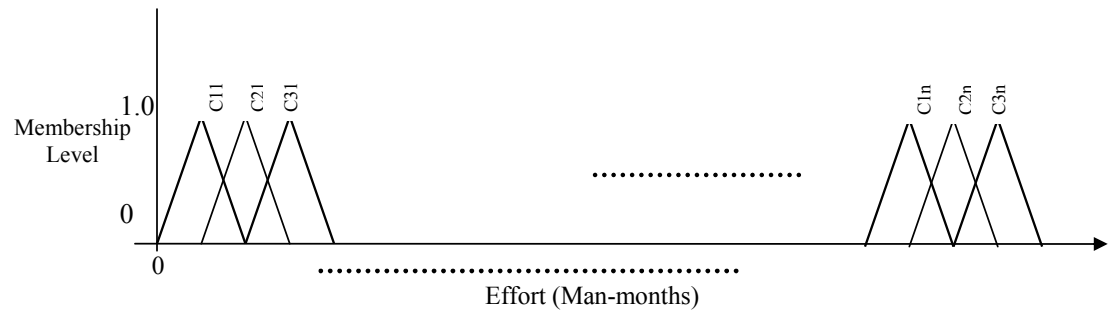
**Figure 13: COCOMO coefficients fuzzification**

In addition, for size of software fuzzification, we have the following form of fuzzy sets (triangular membership function) in the simplest case:



**Figure 14: Fuzzy sets for hypothetical software sizes**

The effort fuzzy sets will correspondingly be:



**Figure 15: Fuzzy sets for hypothetical software efforts**

Our rules formulated, based on the fuzzy sets of modes, sizes and efforts appear in the following form:

*IF mode is organic AND size is s1 THEN cost is c11*

*IF mode is semi-detached AND size is s1 THEN cost is c12*

*IF mode is embedded AND size is s1 THEN cost is c13*

*IF mode is organic AND size is s2 THEN cost is c21*

*IF mode is semi-detached AND size is s2 THEN cost is c22*

*IF mode is embedded AND size is s2 THEN cost is c23*

...

*IF mode is  $m_j$  AND size is  $s_i$  THEN cost is  $c_{ij}$   $(1 \leq i \leq n, 1 \leq j \leq 3)$*

Where  $m_i$ 's are the fuzzy values for the fuzzy variable *mode*,  $s_i$   $(1 \leq i \leq n)$  are the fuzzy values for the fuzzy variable *size*, and  $C_{ij}$   $(1 \leq i \leq n, 1 \leq j \leq 3)$  are the fuzzy values for fuzzy variable cost (effort).

These rules are trained to improve their prediction quality, and the resulting fuzzy inference system (FIS) can be used to predict nominal effort early in the software life cycle, when a detailed knowledge of the cost drivers cannot be ascertained.

#### PART II: Fuzzy Adjustment Factors

In the case of E2, we considered each cost driver differently, since their definitions are based on different criteria. From Equation 7, we can have the following:

$$E21 = (EM_1), E22 = (EM_2), \dots E2n = (EM_n) \quad \textbf{Equation 8}$$

### Cost Drivers Categorization for Fuzzy Logic Implementation

Rating scales of the 15 cost drivers presented in Table 11 are given in Table 12. The corresponding effort multipliers are also given in Table 13. In the effort multiplier table, a rating less than 1 denotes a factor that can decrease the effort, while a rating greater than 1 denotes a factor that extends the effort. Lastly, a rating equal to 1 does not extend nor reduce the effort.

In a consistent effort to aid the fuzzification of the cost drivers, we have identified and categorized the 15 cost drivers (Table 12) into four different classes, resulting from the definitions and rating scales of the drivers and our analysis. The four classes are:

1. Cost drivers expressed using ordinary description - RELY, CPLX, MODP, TOOL
2. Cost drivers given as a range of values - DATA, TURN
3. Cost drivers expressed in percentages/percentiles - TIME, STOR, ACAP, PCAP, SCED
4. Cost drivers expressed in months - VIRT, AEXP, VEXP, LEXP



**Table 12: COCOMO Cost Drivers Ratings**

	Very Low	Low	Nominal	High	Very High	Extra High
DATA		$D/P^4 < 10$	$10 \leq D/P < 100$	$100 \leq D/P < 1000$	$D/P \geq 1000$	
TURN		Interactive	Turnaround < 4 hours	4-12 hours	> 12 hours	
TIME			$\leq 50\%$ use of available execution time	70%	85%	95%
STOR			$\leq 50\%$ use of available storage	70%	85%	95%
ACAP	15 <sup>th</sup> percentile	35 <sup>th</sup> percentile	55 <sup>th</sup> percentile	75 <sup>th</sup> percentile	90 <sup>th</sup> percentile	
PCAP	15 <sup>th</sup> percentile	35 <sup>th</sup> percentile	55 <sup>th</sup> percentile	75 <sup>th</sup> percentile	90 <sup>th</sup> percentile	
SCED	75% of nominal	85%	100%	130%	160%	
VIRT		Change every 12 months	6 months	2months	2 weeks	
AEXP	$\leq 4$ months experience	1 year	3 years	6 years	12 years	
VEXP	$\leq 1$ month experience	4 months	1 year	3 years		
LEXP	$\leq 1$ month experience	4 months	1 year	3 years		
RELY	Effect: slight inconvenience	Low, easily recoverable losses	Moderate, recoverable losses	High financial loss	Risk to human life	
MODP	No use	Beginning use	Some use	General use	Routine use	
TOOL	Basic microprocessor tools	Basic mini tools	Basic midi/maxi tools	Strong maxi programming, test tools	requirements , design, management documentati on tools	
CPLX <sup>5</sup>						

---

<sup>4</sup> **P-Size in KLOC, D- Database size**

<sup>5</sup> The ratings of the CPLX cost driver are based on descriptions in a table structure that can be found in [3].

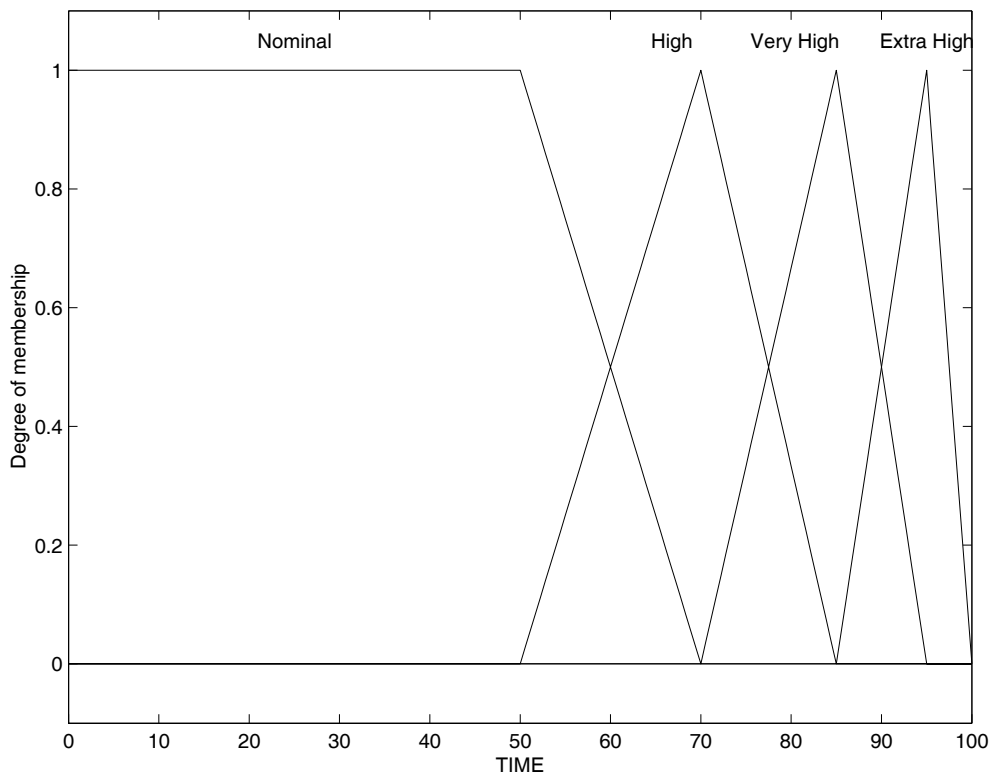
**Table 13: COCOMO Effort Multipliers**

<b>Cost Driver</b>	<b>Very Low</b>	<b>Low</b>	<b>Nominal</b>	<b>High</b>	<b>Very High</b>	<b>Extra High</b>
DATA		0.94	1.00	1.08	1.16	
TURN		0.87	1.00	1.07	1.15	
TIME			1.00	1.11	1.30	1.66
STOR			1.00	1.06	1.21	1.56
ACAP	1.46	1.19	1.00	0.86	0.71	
PCAP	1.42	1.17	1.00	0.86	0.70	
SCED	1.23	1.08	1.00	1.04	1.10	
VIRT		0.87	1.00	1.15	1.30	
AEXP	1.29	1.13	1.00	0.91	0.82	
VEXP	1.21	1.10	1.00	0.90		
LEXP	1.14	1.07	1.00	0.95		
RELY	0.75	0.88	1.00	1.15	1.40	
MODP	1.24	1.10	1.00	0.91	0.82	
TOOL	1.24	1.10	1.00	0.91	0.82	
CPLX	0.70	0.85	1.00	1.15	1.30	1.65

From the four classes identified, it is clear that classes 2-4 present the need for fuzzification. Sensitivity analysis carried out in the next section justifies this claim. However, a few of the cost drivers, like those in Class 1 (appear in grey color in Table 12 and Table 13) specifically, do not lend themselves directly to fuzzy sets definition as the ordinary verbal descriptions are not enough. The availability of experts to give such information that could lead to fuzzy sets representation of Class 1 cost drivers would allow all the cost drivers to be amenable to fuzzification.

For incorporating the cost drivers into our framework, we define fuzzy sets for each linguistic values, “Low”, “Very Low”, etc as it applies to each cost driver. Then rules are formulated using the cost drivers in the antecedent, and their effects on effort in the consequent. Different fuzzy inference systems (FIS) are then developed to model each cost driver. The defuzzified values from each of the FIS are aggregated as EAF to adjust the predicted output from the trained FIS for nominal effort.

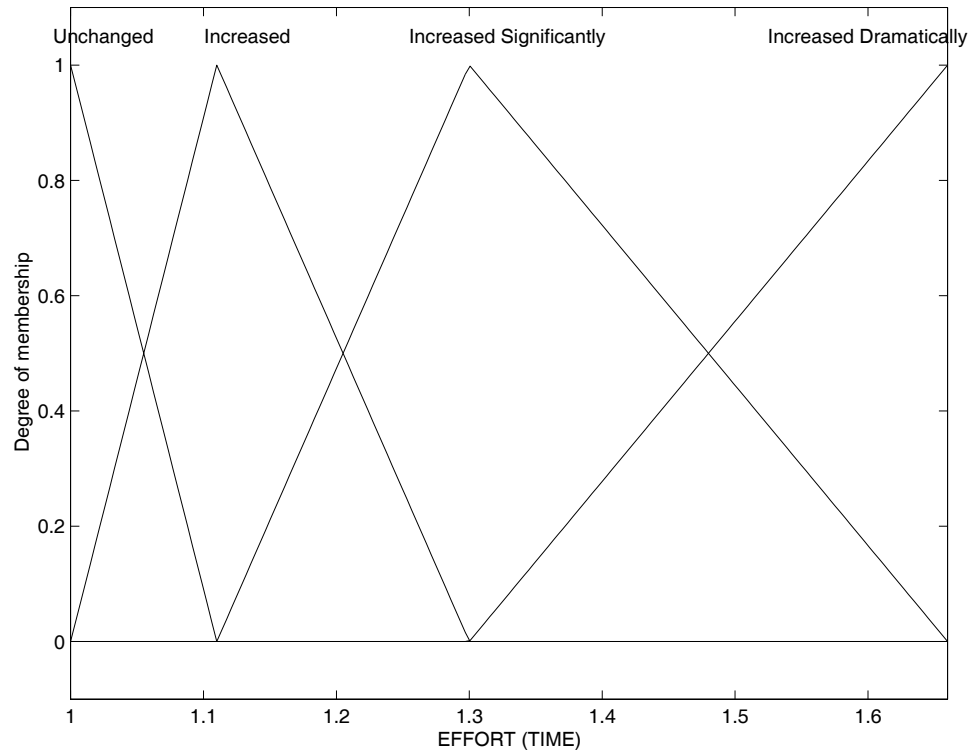
For a sample rule formulation for the cost drivers, as carried out for E1, let us suppose we consider the TIME cost driver, we define fuzzy membership functions of the form given in Figure 16.



**Figure 16: Sample membership functions for the TIME Cost Driver**

As presented for  $PM_{nominal}$ , rules formulation for cost drivers requires fuzzification of the consequent, expressing the effect of the cost drivers on effort. Suppose we consider the TIME cost driver again, we can derive membership functions for the consequents from the effort multipliers in Table 13. Figure 17 gives a hypothetical membership functions definition for such consequent derived from the effort multipliers of the TIME cost driver. The fuzzy sets we have defined for the TIME cost driver in Figure 16 is also hypothetical and derived from the nature of the categorization in the rating

scales, it is however not a static definition since experts have the option of redefining the fuzzy sets as it suits their own development environment.



**Figure 17: Effect of Cost Driver (TIME) on Effort**

Summing the discussion on cost driver TIME into rules, we have the following rules:

*IF TIME is Nominal THEN Effort is Unchanged*

*IF TIME is High THEN Effort is Increased*

*IF TIME is Very High THEN Effort is Increased Significantly*

*IF TIME is Extra High THEN Effort is Increased Dramatically*

In concluding our presentation of the framework, it is worth noting that our rules formulated for E1 ( $PM_{nominal}$ ) is trained using an adaptive training algorithm that we present in Chapter 5. The rules formulated for E2 (cost drivers) do not need any training and are simply developed into fuzzy inference systems. However, the membership functions definition and rules formulation are open to experts' knowledge, because our approach is transparent.

#### 4.3.4 Sensitivity Analysis

In this section, we present a sensitivity analysis on the COCOMO model, to justify why the fuzzification is necessary. Our sensitivity analysis is similar to that presented on the DATA cost driver by Idri and Abran [23] . Our analysis here covers more aspects, as we consider the complete model.

Consider Equation 3, the original COCOMO model

$$\text{Effort} = A \cdot [Size]^B \cdot \prod_{i=1}^{15} EM_i$$

We start our analysis by considering the effects of *size* and *mode* on the main model equation.

## CASE I: SENSITIVITY OF MODE AND SIZE TO USER INPUT

### Fuzziness of Size

Given a semi-detached project of size 200KDSI, effort is calculated with the following equation:

$$\text{Effort} = A \cdot [\text{Size}]^B$$

And we have,

$$\text{Effort} = 3.0 \times [200]^{1.12} = 1133.1 \text{ Man-months} \quad \textbf{Equation 9}$$

If the size was estimated to be 250KDSI, as against 200KDSI (i.e. 25% error), effort becomes 1454.8 Man-months, about 28.4% increase. Depending on the labor rate, the resulting difference in cost can be very large. This would even be more serious when coupled with inaccurate estimation of *mode* and categories of cost drivers.

There is therefore a dire need to represent size with fuzzy sets (so that size may take a “range” shape), such that this great sensitivity to user input is reduced.

### Fuzziness of Mode:

Suppose the project in the example above is actually of size 200 KDSI, but it is more of embedded than organic (e.g. 90% embedded and 10% organic),

the semi-detached constants and exponents should intuitively not apply directly.

Suppose we consider the percentages stated above in our model equation, then the effort predicted would be different from when the scaling factors for semi-detached mode is used directly. An example interpolation is given in Table 14 to show the need for us to capture the imprecision in mode using fuzzy sets:

**Table 14: An Illustration to mimic the Fuzziness of mode using a sample project**

<b>Semi-Detached Project (90% Embedded, 10% Organic)</b>		
<b>Organic (10%)</b>	$A1 = 3.2 \times 0.1$	$B1 = 1.05 \times 0.1$
<b>Embedded (90%)</b>	$A2 = 2.8 \times 0.9$	$B2 = 1.2 \times 0.9$
<b>Semi-Detached Project (new constants &amp; exponents)</b>	$A = A1 + A2$ $A = 2.84$	$B = B1 + B2$ $B = 1.185$

From the table above, we will consider  $A = 2.84$  and  $B = 1.185$ , instead of  $A = 3.0$  and  $B = 1.12$  that would generally be assumed for any semi-detached project. We have the new effort calculated as:

$$\text{Effort} = 2.84 \times [200]^{1.185} = 1513.7 \text{ Man-months}$$

**Equation 10**



Equation 9 and Equation 10 are supposed to evaluate the same software project of 200KDSI, but the fuzziness of the semi-detached category was not taken into consideration in Equation 9 (as we have explained using the example above) and we have effort reported to be different and less by 25.1%.

Ignoring the fuzziness in size of project and mode of development, results in a model that ignores the imprecision in user input.

#### **CASE II: SENSITIVITY OF THE COST DRIVERS**

The effort multipliers resulting from the cost drivers' ratings have very great effects on the final effort estimate and eventually the cost, depending on the labor rate. Choosing maximum possible ratings for all the cost drivers requires effort to be multiplied by 73.4 (a large increase), while choosing lowest possible values for all the 15 cost drivers requires the nominal effort to be multiplied by 0.097.

Similar to the sensitivity analysis presented on the DATA cost driver in [23], we give an example to justify the need for fuzzification of the cost drivers. From Table 1, no project can occupy more than one category. Consider the DATA cost driver for instance,  $D/P = 999.99$  and  $D/P = 100.1$  will have the same effort multiplier, as DATA is rated High in both cases. Meanwhile,

1000.01 is rated as Very High, with different effort multiplier than  $D/P = 999.09$ .

Thus, two analogous projects P1 and P2 with  $D/P = 999.99$  and 1000.01 respectively, will rate P1 as HIGH and P2 as VERY HIGH. The predicted effort is multiplied by 1.08 for P1 (8% increment) while that of P2 is multiplied by 1.16 (16% increment). It becomes more serious if this happens consistently for all the cost drivers' ratings.

In the same vein, the TIME cost driver, for example, discussed the ratings:  $\leq 50\%$  use of available execution time, 70% use of execution time, but no information about what happens between 51%-69%. A project P having a percentage use of execution time to be 50% would not have its nominal effort affected by the TIME cost driver while a project rated on the 70% scale would be increased by 11%. The failure to address the procedure to follow in choosing multipliers for projects that the use of available execution time falls between 51% – 69% would affect the accuracy of effort predicted. Fuzzification of these cost drivers will answer these ambiguities (see Figure 17). Specific analysis has not been given here as we did for *size* and *mode*, since the same argument goes for the cost drivers as well.

#### 4.3.5 COCOMO II and the Framework

COCOMO II consists of three models: Applications Composition, Early Design, and Post Architecture. Boehm *et al.* [5] stated that, the post architecture model of COCOMO II is the closest in form to the intermediate version of COCOMO '81. COCOMO II post architecture model is given as:

$$\text{Effort} = A \times [\text{Size}]^{B + \sum_{i=1}^5 SF_i} \times \prod_{i=1}^{17} EM_i \quad \text{Equation 11}$$

Where A is a constant, B is an exponent for economies/diseconomies<sup>6</sup> of scale and SF<sub>i</sub>'s are scaling factors defined in similar manner to the effort multipliers. For a detailed discussion of COCOMO II, the reader is referred to Boehm *et al.* [5]

In order to show the linear independence of some of the input parameters to COCOMO II model, in line with similar evaluation we discussed earlier for COCOMO '81, we take the logarithms of both sides of Equation 11, to have:

$$\ln(\text{Effort}) = \ln(A) + B \cdot \ln(\text{Size}) + \sum_{i=1}^5 SF_i \ln[\text{Size}] + n \left( \prod_{i=1}^{17} EM_i \right) \quad \text{Equation 12}$$

This implies that the model in Equation 12 can also follow the approach in our framework as discussed for the original COCOMO model. However,

---

<sup>6</sup> Economies/Diseconomies of scale is explained in [13]

using COCOMO II model will incorporate more features that are subjective. For example, automatic data generation for training will not be straightforward, because of the scale factors  $SF_i$  (whose definition is based on rating scales (i.e. nominal, high, etc) like the cost drivers).

Beside the constraint above, COCOMO II is not yet well established like the original model, and the calibration is still in progress as more data are being collected to update the COCOMO II database.

For this reasons, we would not be addressing COCOMO II further in this thesis research. However, the good news remains that, our approach is still valid and applicable to COCOMO II once experts are available to provide the information required for membership function formulation.

## ***CHAPTER 5***

### **TRAINING**

#### **5.1 Introduction**

This chapter presents the training approach employed in our framework to optimize the developed fuzzy inference system. The chapter begins with a brief discussion of learning approaches in neuro-fuzzy systems in general, with particular reference to the Generic Fuzzy Perceptron. This is followed by the formulation of our training task in light of the fuzzy perceptron. Finally, the training algorithms are presented.

#### **5.2 Training Approaches in Neuro-Fuzzy Systems**

In Chapter 2, we introduced adaptive fuzzy systems as fuzzy systems whose parameters are adjusted based on numerical information. The structure of a fuzzy system is given by its rules and the number of fuzzy sets used to partition each variable. The parameters of any fuzzy system consist of the rules, the shapes and locations of the membership functions (MFs) of such fuzzy sets. For an adaptive fuzzy system, however, the only parameters that change during training are the fuzzy sets. Before the parameters of a fuzzy

system can be adapted in a training process, the structure of the rulebase must be determined.

The basic design problems in a fuzzy system, therefore, are the choice of appropriate fuzzy rules, membership functions of the fuzzy sets and tuning of both to improve performance. One way to tune the parameters of a fuzzy system based on training data is to use learning methods that are derived for neural networks. This hybridization of neural networks and fuzzy logic is the basic idea behind the neuro-fuzzy system introduced in Section 3.5.4.1. The next section discusses two approaches for building adaptive fuzzy systems using neuro-fuzzy hybridization.

### **5.2.1 Neuro-Fuzzy Hybridization**

Neuro-fuzzy hybridization is done in two ways [39]: fuzzy neural networks (FNN) and Neuro-Fuzzy systems (NFS). FNN is a neural network equipped with the capability of handling fuzzy information [28]. NFS is a fuzzy system augmented by neural networks to enhance some characteristics like flexibility and adaptability [47][48][45]. NFSs are particularly of interest in our training since they realize the process of fuzzy reasoning, where the connection weights of the network correspond to the parameters of fuzzy reasoning. An example of a neuro-fuzzy model is the generic fuzzy perceptron by Nauck [47].

### 5.2.2 The Generic Fuzzy Perceptron

Fuzzy Perceptron is a generic model of feed forward multilayer perceptron that uses fuzzy sets as weights. The weights of the network are membership functions that represent the linguistic values of the input variables and output variables of a fuzzy logic system, respectively. It was first proposed by Nauck [47] and further discussed by Nauck *et al.* [45] as an architecture upon which training algorithms for fuzzy rules can be built. The fuzzy perceptron architecture eliminates the black-box problem that results from using neural network model directly to train fuzzy systems.

The generic model is based on a 3-layer fuzzy perceptron, consisting of input layer, rules layer and output layer. The input layer is connected to the rules layer by weights represented as fuzzy sets ( $\mu_i^{(j)}$ ) of linguistic variable of the antecedent, and the rules layer is connected to the output layer by fuzzy sets ( $\nu_k$ ) of linguistic variable of the consequent. The structure of the 3-layer fuzzy perceptron with 2 input variables ( $\xi_1, \xi_2$ ) in the input layer, 5 rules ( $R_l$ ) and 1 output variable ( $\eta$ ) is given in Figure 18.

From Figure 18,  $\mu_1^{(1)}$  denotes the first fuzzy set of input variable 1,  $\mu_2^{(1)}$  denotes second fuzzy set of input variable 1 and so on. Similarly,  $\mu_1^{(2)}$  denotes the first fuzzy set of input variable 2,  $\mu_2^{(2)}$  denotes the second fuzzy set of input variable 2, etc.

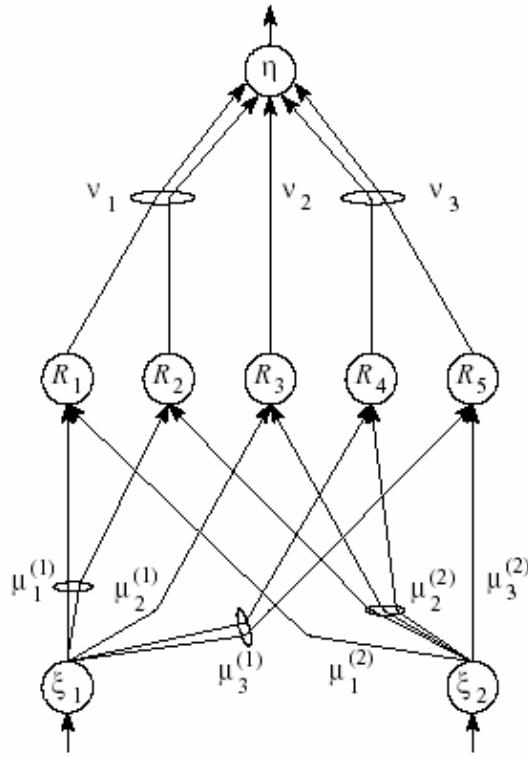


Figure 18: Three-Layer Generic Fuzzy Perceptron [47][45]

#### 5.2.2.1 Rules Learning in Fuzzy Perceptron

A heuristic learning procedure was derived for the fuzzy perceptron by Nauck [47]. The heuristic is similar to the back-propagation algorithm for neural networks discussed by Michael Negnevitsky [49] and Nauck *et al.* [45]. It can be adapted to problems that can be modeled as fuzzy systems. It uses fuzzy sets for weights and fuzzy error. The steps of the algorithm are:

1. Choose a training sample and propagate the input vector across the network to get the output.



2. Determine the error in output, and the error gradient in all the other layers
3. Determine the parameter changes for the fuzzy weights and update the fuzzy weights
4. Repeat until the fuzzy error is sufficiently small after an epoch is complete.

#### **5.2.2.2 Fuzzy Perceptron Applications**

Variants of the generic fuzzy perceptron architecture and the learning algorithm have been implemented for different type of problems modeled as neuro-fuzzy systems. Notable of these models are: neuro-fuzzy control model (NEFCON) [46][45], neuro-fuzzy classification model (NEFCLASS) for data classification [44][43][45], and neuro-fuzzy function approximation model (NEFPROX) [45].

NEFCON, for example, uses a learning model built upon the fuzzy perceptron architecture. The learning process is divided into two phases: The first phase learns an initial rulebase, if no prior knowledge of the system is available; the second phase optimizes the rules by shifting or modifying the fuzzy sets of the rules.

Three methods to learn an initial rulebase were identified:

1. Methods starting with an empty rulebase ([44])
2. Methods starting with a full rulebase (combination of every fuzzy set in the antecedents with every conclusion) ([42])
3. Methods starting with a random rulebase ([33])

The first two methods were employed in developing NEFCON.

In the second set of methods for the full rulebase, all possible combinations of the input and output fuzzy sets are carried out for rules formulation. For example, if we have 2 inputs with 5 fuzzy sets (linguistic variables) each, and 1 output with 5 fuzzy sets, the number of initial rules created will be  $5^3=125$ . The training of the fuzzy inference system (FIS) with these rules is done in two stages. The first stage trains the rules developed using this second set of methods to discover and remove unnecessary rules. At the end of this training stage, the FIS with the appropriate rules is developed. In order to improve the prediction quality of the developed FIS, the second stage involves a further training of the resulting rulebase using fuzzy set learning algorithms to modify/shift the rules fuzzy sets.

### 5.3 Effort Prediction Framework – The Training Approach

Our strategy requires building learning capabilities into our fuzzy logic framework so that the system can learn the importance of the input features and their relationships with effort. The process of fuzzification discussed in Chapter 4 acts as the starting point for training. Our training approach for the framework is two-stage - the basic (nominal effort) part of the model is trained independently, while the cost drivers are fuzzified and developed into another independent fuzzy inference systems.

The learning model used for the basic part of the model is based on the generic fuzzy perceptron architecture. Our model to train the basic part of the framework is similar in structure to NEFCON model (since it has one output feature), but similar in its learning approach to the NEFPROX's model that uses supervised learning as opposed to the reinforcement learning used in NEFCON. For further discussion of these learning paradigms (i.e. reinforcement and supervised learning), the reader is referred to Rich and Knight [56], Russell and Norvig [57], or Jang *et al.* [26].

### 5.3.1 Fuzzy Reasoning in the Nominal Effort Model Using Fuzzy Perceptron

We consider once again the nominal effort component of the framework for effort model. Our rules formulation discussed in Chapter 4 follows the Mamdani max-min fuzzy reasoning discussed in Chapter 2, and the reasoning can be transformed to fuzzy perceptron structure in a straightforward manner.

We have generated rules of the form:

$$\text{IF } MODE \text{ is } M_j \text{ AND } SIZE \text{ is } S_i \text{ THEN } EFFORT \text{ is } C_{ji} \quad \text{Equation 13}$$

Using the Mamdani max-min inference system to evaluate the complete set of rules, the cost derived from Equation 13 will be given as:

$$C_{ji}(effort) = \max_j \{ \min \{ M_j(x), S_i(y) \} \} \quad \text{Equation 14}$$

Where,

$$M_j(x) : \mathfrak{R} \rightarrow [0,1], \text{ and also } S_i(y) : \mathfrak{R} \rightarrow [0,1] \text{ (see Figure 3).}$$

Equation 14 is the fuzzy reasoning for evaluating a rulebase to get the predicted output. In order to train the rulebase and still preserve the meaning of the relationship that exists between the variables, the fuzzy reasoning is represented using a fuzzy perceptron learning structure.

Suppose we derive the following three rules using Equation 13:

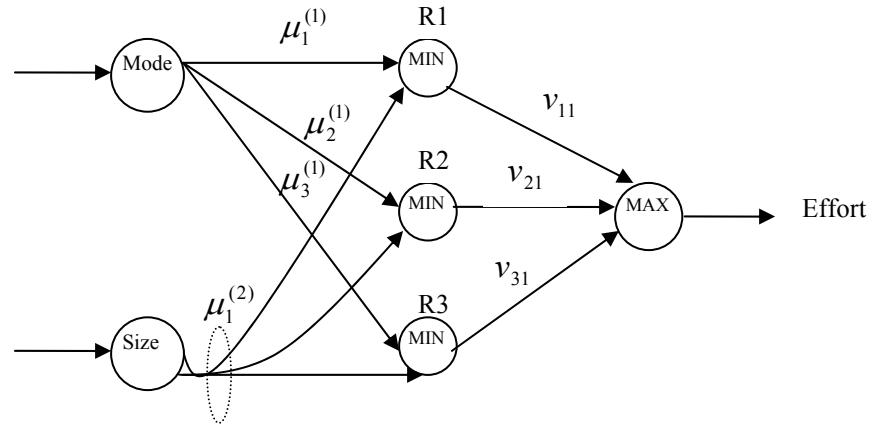
RULES:

R1: IF *MODE* is  $M_1$  AND *SIZE* is  $S_1$  THEN *EFFORT* is  $C_{11}$

R2: IF *MODE* is  $M_2$  AND *SIZE* is  $S_1$  THEN *EFFORT* is  $C_{21}$

R3: IF *MODE* is  $M_3$  AND *SIZE* is  $S_1$  THEN *EFFORT* is  $C_{31}$

the fuzzy perceptron representation of the rules using the Mamdani fuzzy reasoning system is given in Figure 19:



**Figure 19: Cost Model Rules Represented using Fuzzy Perceptron Structure**

$\mu_i^{(1)}$ ,  $\mu_i^{(2)}$  are membership functions of the *mode* and *size* input variables respectively, while  $v_{ij}$ 's are fuzzy sets of the effort output variable.  $M_j, S_i$

and  $C_{ji}$  are linguistic terms. Considering rule R1, for instance,  $M_1$ ,  $S_1$  and  $C_{11}$  are linguistic terms represented by the fuzzy sets  $\mu_1^{(1)}$ ,  $\mu_1^{(2)}$  and  $v_{11}$  respectively.

From the representation above, it is clear that the architecture of the neuro-fuzzy perceptron is determined by the rules and the fuzzy sets of the underlying problem.

### 5.3.2 Training Algorithms for the Effort Prediction Framework

We built our training algorithm on the generic fuzzy perceptron architecture, adapting the algorithms as appropriate for our problem.

The training implementation involves these steps:

1. Structure-Learning Phase: for building the IF-THEN rules using the knowledge built into the COCOMO model
2. Parameter-Learning Phase (for tuning the membership functions and rules to optimize)

#### 5.3.2.1 Structure Learning Phase - Building the Rulebase

The structure-learning phase involves building a full rulebase to be optimized during training of the membership functions. Our proposed approach for building a rulebase of the fuzzy inference system for nominal

effort prediction is a special case of the five-step procedure proposed by Wang and Mendel [72] for generating fuzzy rules from numerical data pairs.

An outline summary of the five-step procedure by Wang and Mendel is given as follows:

- Divide the input and output spaces into fuzzy regions
- Generate fuzzy rules from given data pairs
- Assign a degree to each rule
- Create a combined fuzzy rule base
- Determine a mapping based on the combined fuzzy rule base

Since our rules formulation mechanism is guided by the underlying model (COCOMO), we will only adopt and modify Step I of Wang and Mendel for partitioning the *size* input variable into fuzzy regions using fuzzy sets. The prior knowledge available makes the remaining steps not applicable to our approach. The steps we follow to build the initial FIS are described by Algorithm 1.1.

### ALGORITHM 1.1 – Building initial Fuzzy Inference Matrix

#### STEP 1: Defining the input variables membership functions

1. Define fuzzy sets for *mode* input variable (intuitive from *mode* classification) using triangular membership function (see Figure 12), or any other shape of membership functions that is applicable, like Gaussian MFs.
2. For the *size* input variable, suppose the domain interval is  $[s^-, s^+]$  e.g. (1 – 100KDSI), where the domain interval means that most probably the size variable will assume values that lie in this interval. Divide the domain interval into  $2N+1$  region, and assign each region a fuzzy membership function. Figure 20 shows an example where the domain interval is divided into five regions ( $N = 2$ ). The shape of each membership function is triangular in this case – one vertex lies at the center of the region and has membership degree value of 1; the other two vertices lie at the centers of the two neighboring regions, respectively, and have membership degree values equal to zero. That is, each MF is defined as  $TMF(\alpha, m, \beta)$ , with center  $m$  and support  $[\alpha, \beta]$ . The membership degree in TMF of center  $m$  is 1, and those of  $\alpha$  and  $\beta$  are 0.

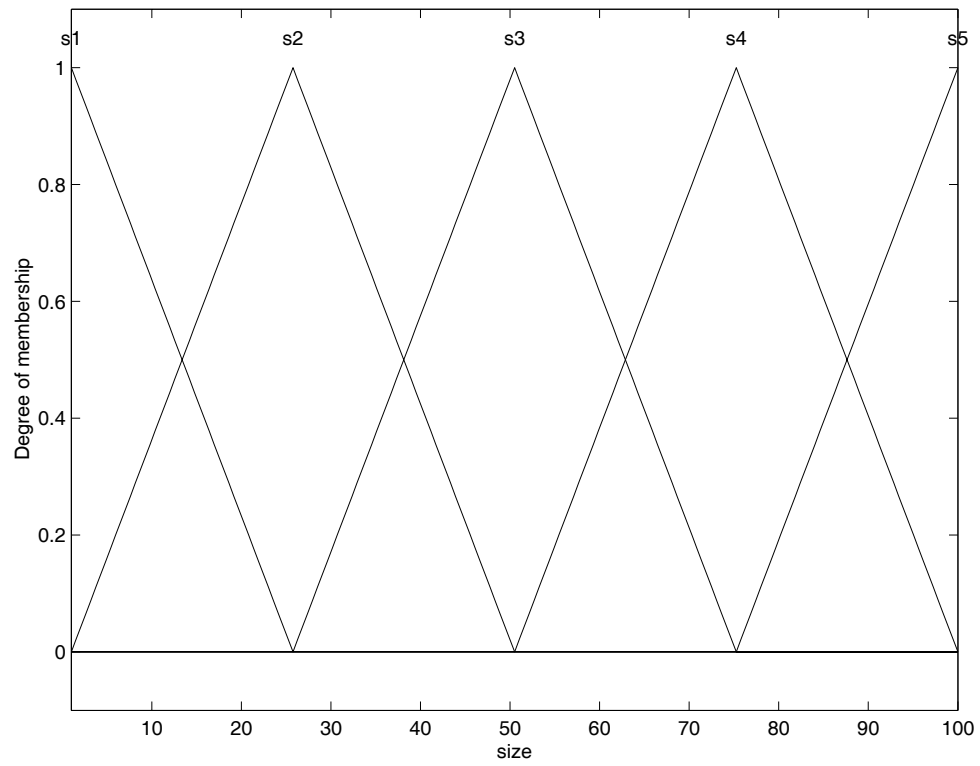
#### STEP 2: Defining the output variables membership functions

1. For the output variable, *effort*, the parameters of each MF of a selected *size* MF in Step 1 is plugged into COCOMO nominal effort model to calculate the parameters of the corresponding *effort*, for each of the three *modes*. This implies that, for every *size* MF, we will have three different membership functions corresponding to three regions.
2. Repeat (1.) as many as the number of input MF we have for *size*. A sample output MF based on the (1-100KDSI) is given in Figure 21.

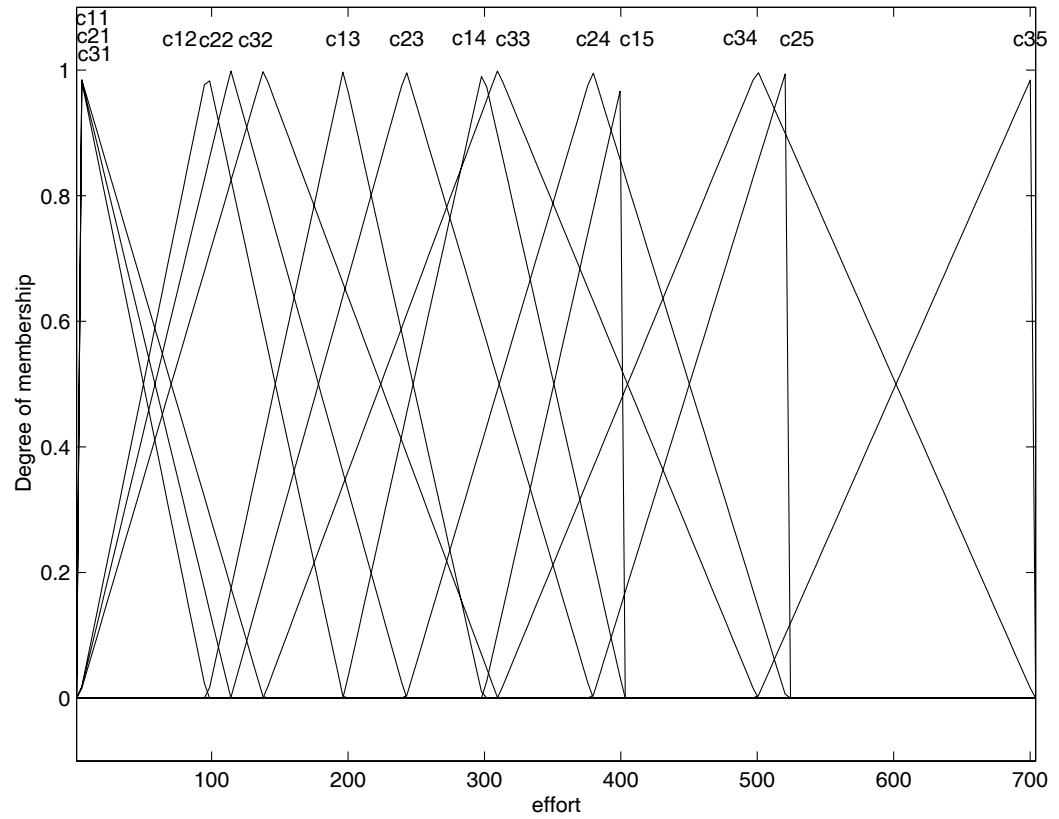
#### STEP 3: Formulating the rules and populating the rulebase

Using the prior knowledge embedded in COCOMO model, we formulate a rule reflecting the relationship between corresponding *mode*, *size* and *effort* MFs selected. This is repeated for as many as the number of *effort* MFs created in Step 2. Thus, we will have  $3(2N + 1)$  regions and MFs in the output. Similarly, we will have  $3(2N + 1)$  fuzzy rules in the rulebase of the fuzzy system





**Figure 20: Divisions of the input variable **SIZE** into fuzzy regions and the corresponding membership functions**



**Figure 21: The membership functions of output variable EFFORT based on the SIZE MFs of Figure 20.**

Algorithm 1.1 is guided by the prior knowledge already embedded in the cost model, which avoids the computational overhead of searching for spurious rules after the rulebase is created. This gain emphasizes the importance of prior knowledge, unlike NEFCON that generates all possible combinations of antecedents and consequent fuzzy sets for rules development. Suppose we have fuzzy effort prediction system that is composed of 2-input 1-output

with 3 and 5 fuzzy sets for the two antecedents respectively, and 5 fuzzy sets in the consequent, we will have 15 rules as against NEFCON's  $3 \times 5 \times 5 = 75$  fuzzy rules. Our rules are reflective of the actual relationships between the variables, and we proceed to the membership functions learning algorithms discussed in the next section, instead of looking for heuristics to identify spurious rules before MF training.

#### **5.3.2.2 Parameter Learning Phase**

The objective of the parameter-learning phase is to adjust parameters of the fuzzy inference system (FIS) such that the error function during training using the training dataset, reaches minimum or is less than a given threshold [48][63]. The error measure is not only used to guide the learning process, but also to evaluate the performance of the final model. Training is achieved by adapting the parameters of the membership functions and rules in the input/output layers.

In order to train the membership functions we need to generate training dataset. Algorithm 1.2 describes our approach to generating artificial dataset for training and validation.

**ALGORITHM 1.2** – Generate artificial datasets for training and validation

- 1: Generate random numbers for a desired number, say  $K$ , of unique sizes in the domain interval  $[s^-, s^+]$  considered in Algorithm 1.1.
- 2: For every number generated in (1), randomly select one of the three development modes and calculate corresponding effort value using the nominal effort model .
- 3: Repeat 1 and 2 until  $K$  data points have been generated, with each data pattern consisting of values for *size* and *mode* as input, and *effort* as target.
- 4: Partition the  $K$  data points into training and validation datasets. The training datasets consist of two-third of the entire dataset while the remaining one-third is left for testing after training.

The algorithms for fuzzy set learning in a Mamdani-type fuzzy system, originally presented in [47] and [48] follow the four-step procedure discussed in Section 5.2.2.1. We have modified this algorithm where applicable to suit our problem. The modification of the MFs of rules is based on the extent of contribution of each rule to the output.

The fuzzy set learning algorithms (Algorithms 1.3 – 1.6) use the following notations:

- $L$ : a set of training data with  $|L| = s$ , where patterns  $p \in R^n$  as input is mapped to a target  $t \in R$
- $(p, t) \in L$ : a training pattern consists of an input vector  $p \in R^n$  and target  $t \in R$
- $A_r = (\mu_r^{(1)}, \dots, \mu_r^{(n)})$ : the antecedent of rule  $R_r$ .
- $A_r(p)$  denotes the degree of fulfillment of rule  $R_r$  (with antecedent  $A_r$ ) for pattern  $p$ , i.e.  $A_r(p) = \min\{\mu_r^{(1)}(p_1), \dots, \mu_r^{(n)}(p_n)\}$ .

- $\mu_r^{(i)}$  : a fuzzy set of input variable  $x_i$  ( $i \in \{1, \dots, n\}$ ), ( $x_1 = mode$ ,  $x_2 = size$ ) that appears in the antecedent of fuzzy rule  $R_i$  ( $r \in \{1, \dots, k\}$ ).
- $\nu_r$  : a fuzzy set of output variable  $y$  (effort) that appears in the consequent of fuzzy rule  $R_r$ .
- $\delta$  : is the learning rate of the training algorithm.

The training algorithm is presented in four parts in Algorithms 1.3 – 1.6. Algorithm 1.3 implements the main loop of the training procedure. In each loop, the algorithm propagates a training pattern, determines the output of the fuzzy system, and computes the parameter updates of the consequent and of the antecedent MFs using the error value determined at the output side of the fuzzy system, as outlined in section 5.2.2.1.

The main information derived from the error value is whether the contribution of a fuzzy rule to the overall output values should be increased or decreased. The error value is used to compute the delta of the modification. Meanwhile, the actual modification of the consequent and antecedent fuzzy sets is based on some heuristics in the algorithms.

The learning algorithms are presented below:

**ALGORITHM 1.3** - Fuzzy set training in a Mamdani-type fuzzy system

```

1: repeat
2:   for all patterns  $(\mathbf{p}, \mathbf{t}) \in L$  do
3:     propagate the next training pattern  $(\mathbf{p}, \mathbf{t})$ ;
4:      $E = (t_j - o_j) / \text{outputRange}$ 
5:     for each rule  $R_r$  with  $A_r(\mathbf{p}) > 0$  do   (* Consider only the rules that fire *)
6:       for all  $v_r$  do                         (* modify all output fuzzy sets *)
7:         ComputeConsequentUpdates( $v_r, E, A_r(\mathbf{p}), t$ ); (* Algorithm 1.4 *)
8:         Update( $v_r$ );                          (* Algorithm 1.6 *)
9:       end for
10:       $E_r = (2 \cdot v_r(t) - 1) \cdot |E|$ ;      (* Rule Error *)
11:      (* modify antecedent MF with min. membership degree of all MFs *)
12:       $j = \min_{i \in \{1, \dots, n\}} \{\mu_r^{(i)}(p_i)\}$ ;
13:      ComputeAntecedentUpdates( $\mu_r^{(j)}, E_r, p_j$ ); (* Algorithm 1.5 *)
14:      Update( $\mu_r^{(j)}$ );                        (* Algorithm 1.6 *)
15:    end for   (* end for each rule *)
16:  end for   (* end for all training patterns *)
17:  Calculate RMSRE and test for end criterion
18: until   (* end for each epoch *)

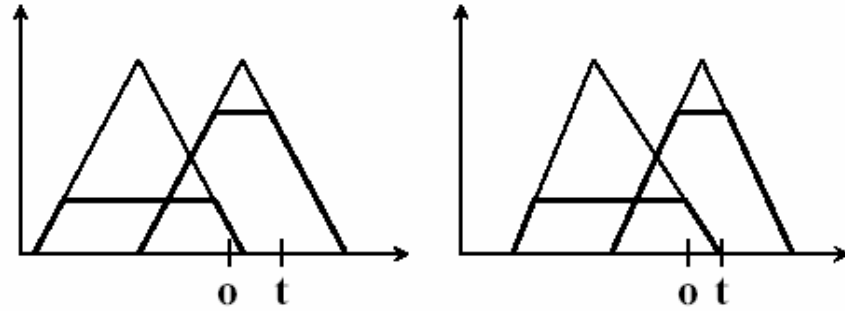
```

From Algorithm 1.3 above, we have normalized the error  $E$  in output, dividing it by the range of the domain interval of the output variable as given in Line 4. The idea behind our normalization is to tolerate small errors, and make the value of the error relative to the value of the output variable. This is necessary because training the fuzzy sets takes the ranges of the individual variables into account. If the output variable is using very large values, the

size of delta computed is equally so large. In order to be completely independent from the ranges of all the variables, we use the normalization of the error. The training algorithm would therefore successfully operate on any range of input/output variables in the dataset, and there would be no need to normalize the dataset between  $[0, 1]$  before training as normally done in neural networks training. This means that the error is bounded and not overemphasized for large errors in output.

The delta of the consequent fuzzy sets are computed by using the *error*,  $E$ , in predicted output for *effort*.

The heuristic for modifying the output of a fuzzy system takes the defuzzification procedure into consideration (e.g. MOM, COA discussed in Chapter 2). To move the output value of a fuzzy system closer to the target value the support of a consequent fuzzy set must be shifted such that the center of the fuzzy sets moves closer to the target value. If the target has non-zero membership with the fuzzy set to be modified, then the support of this fuzzy set is also reduced to focus the fuzzy set on the target value. If the target has zero membership with a fuzzy set to be modified, then the support of this fuzzy set is extended towards the target value. Figure 22 gives an example of modification for two triangular membership functions in the consequent [48].



**Figure 22:** To move an output value  $o$  of a fuzzy system closer to a current target value  $t$  the consequent fuzzy sets are moved towards  $t$ .

Correcting the output of a fuzzy system requires that the degrees of fulfillment (i.e. firing strength) of the considered rules be modified such that we increase the contribution of rules where a consequent fuzzy set yields a high degree of membership for the target value. If the target value has only a small degree of membership with the consequent fuzzy set, the influence of the corresponding rule must be decreased.

In order to update the parameters of an antecedent MFs using the delta values, we need to know the *rule error*,  $E_r$ , of a fuzzy rule. The *rule error* of  $R_r$ , given in *Line 10*, obtains the following error signal from its output variable:

$$E_r = (2 \cdot V_r(t) - 1) \cdot |E|$$

**Equation 15**



Where  $t$  is the target output,  $V_r$  is the output fuzzy set,  $E$  is the error in output of the current pattern, and  $V_r(t)$  is the degree to which the target output belongs to the output fuzzy set.

If  $V_r(t) > 0.5$  then the rule is assumed to be strongly in support of the current target output and we obtain a positive error signal from Equation 15 in order to increase the contribution of the rule (i.e. increasing the rule fulfillment). However,  $V_r(t) < 0.5$  shows the rule is not strongly in support of the current target output, and the error signal from the output of Equation 15 is negative in order to decrease the degree of fulfillment of rule  $R_r$ .

In the original learning algorithm discussed in [48], the rule error (Equation 15) also includes the multiplicative factor  $\{t_r (1 - t_r)\}$  that considers the rule fulfillment,  $t_r$ . This factor generates a rule error that makes sense when the degree of fulfillment is either 0 or 1, but not same for values between 0 and 1. We have eliminated this factor to increase predictive accuracy of the training algorithm within the whole range.

In *Line 5*, we only consider the rules that have degree of fulfillment greater than zero of all the rules in the rulebase. A rule having degree of fulfillment equal to zero does not contribute in anyway to the current output of the fuzzy system.

*Line 11 – 14* of the algorithm shows that we only modify the antecedent fuzzy sets that yield the minimum membership degree ( $\mu_r^{\min}$ ) for the current input pattern. This particular fuzzy set determines the degree of fulfillment of the firing rule, using the Mamdani Min-Max reasoning discussed in Section 2.3.1.2 for fuzzy inference procedure. Updating only MFs with  $\mu_r^{\min}$  is motivated by the idea to keep changes to the fuzzy systems focused on MFs that determines the contribution of each rule.

Algorithms 1.4 – 1.6 are used for parameter updates. In our implementation, we have used triangular memberships function given by:

$$\mu_{a,b,c}(x) = \begin{cases} \frac{x-a}{b-a} & \text{for } x \in [a, b] \\ \frac{c-x}{c-b} & \text{for } x \in [b, c] \\ 0 & \text{otherwise} \end{cases}$$

With  $a \leq b \leq c$

**ALGORITHM 1.4** - Compute updates for consequent fuzzy sets

```

ComputeConsequentUpdate ( $v, e, \tau, t$ )
  (*  $v$ : fuzzy set for which parameter updates must be computed *)
  (*  $e$ : error value *)
  (*  $\tau$ : degree of fulfillment of the rule that uses  $v$  in the consequent *)
  (*  $t$ : current target value from the domain of  $v$  *)
  (*  $a, b, c$  are parameters of the fuzzy set  $v$  *)

1: if ( $v$  is triangular) then
2:    $\text{delta} = \sigma \cdot e \cdot (c - a) \cdot \tau \cdot (1 - v(t));$ 
3:    $\Delta b = \text{delta}$ 
4:   if  $v(t) > 0$  then      (*  $t \in \text{support}(v)$ , focus  $v$  on  $t$  *)
5:      $\Delta a = \sigma \cdot \tau \cdot (b - a) + \text{delta}$ 
6:      $\Delta c = -\sigma \cdot \tau \cdot (c - b) + \text{delta};$ 
7:   else      (*  $t \notin \text{support}(v)$ , shift  $v$  to cover  $t$  *)
8:      $\Delta a = \text{sgn}(t - b) \cdot \sigma \cdot \tau \cdot (b - a) + \text{delta};$ 
9:      $\Delta c = \text{sgn}(t - b) \cdot \sigma \cdot \tau \cdot (c - b) + \text{delta};$ 
10:  end if
11: end if

```

**ALGORITHM 1.5** - Compute antecedent fuzzy set updates

```

ComputeAntecedentUpdates( $\mu, e, p$ )
  (*  $\mu$ : fuzzy set for which parameter updates must be computed *)
  (*  $e$ : rule error value *)
  (*  $p$ : current input value from the domain of  $\mu$  *)
  (*  $a, b, c$  are parameters of  $\mu$  *)

1: if ( $e < 0$ ) then      (* take degree of membership into account *)
2:    $f = \sigma \cdot \mu(p)$     (*  $\sigma$  is a learning rate *)
3: else
4:    $f = \sigma(1 - \mu(p))$ 
5: end if
6: if ( $\mu$  is triangular) then
7:    $\text{delta} = f \cdot e \cdot (c - a) \cdot \text{sgn}(p - b);$ 
8:    $\Delta a = -f \cdot e \cdot (b - a) + \text{delta};$   (*lower bound of support*)
9:    $\Delta c = f \cdot e \cdot (c - b) + \text{delta};$   (*upper bound of support*)
10:   $\Delta b = \text{delta};$ 
11: end if

```

**ALGORITHM 1.6** - Carry out the update of a fuzzy set

**Update** ( $\mu, k$ )  
 (\*  $\mu$ : fuzzy set for which parameter updates must be computed \*)  
 (\*  $k$ :  $k = 1$  for online learning \*)

- 1: **if** (the update of  $\mu$  conflicts with the constraints for  $\mu$ ) **then**
- 2:     modify the updates of  $\mu$  such that the constraints are satisfied;
- 3:     for example the overlap constraints and valid parameters constraints.
- 3: **end if**
- 4: **for all** parameters  $w$  of  $\mu$  **do**
- 5:      $w = w + \Delta w$ ;     (\* apply parameter updates \*)
- 6:      $\Delta w = 0$ ;     (\* reset parameter updates \*)
- 7: **end for**

While Algorithms 1.4 – 1.5 generate the extent of modifications of the parameters of antecedents and consequents fuzzy sets, the actual update is carried out by Algorithm 1.6 that checks to make sure some specified constraints are satisfied.

Some constraints we have implemented include the following:

1. *Valid Parameters*: Makes sure that no fuzzy set is modified such that the condition  $a \leq b \leq c$  does not hold anymore. In addition, no fuzzy set must be modified such that it passes the domain interval of the input/output variable under consideration.
2. *Overlap Constraint*: This makes sure that gaps are not present between adjacent fuzzy sets after modification. That is, adjacent fuzzy sets must always overlap.

## ***CHAPTER 6***

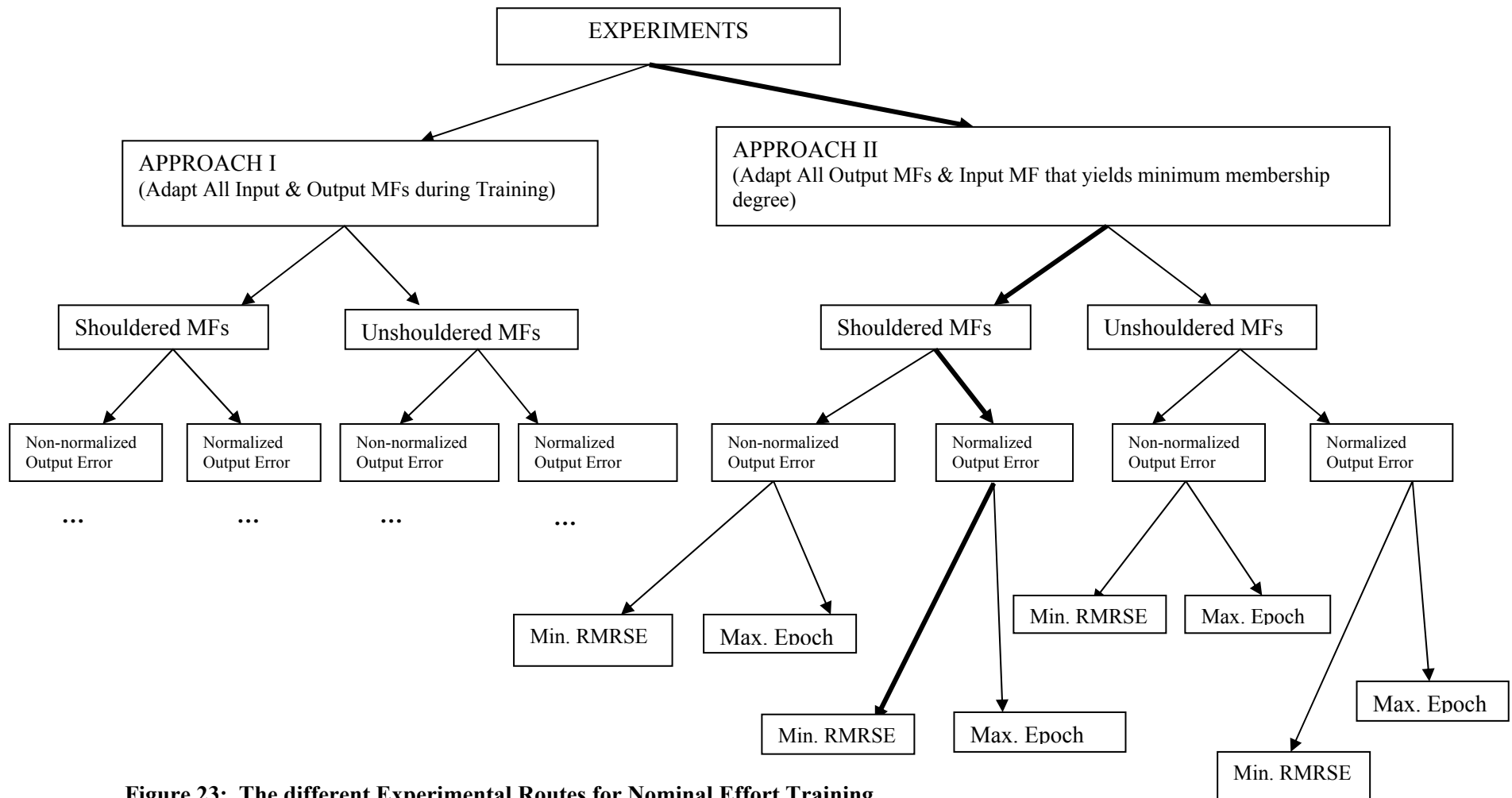
### **EXPERIMENTS AND RESULTS**

#### **6.1 Introduction**

This chapter presents the different routes we have followed in our experiment to realize the learning algorithms discussed in Chapter 5. The chapter also discusses the implementation of the cost drivers. Finally, we present experimental results, discussion and comparison of results.

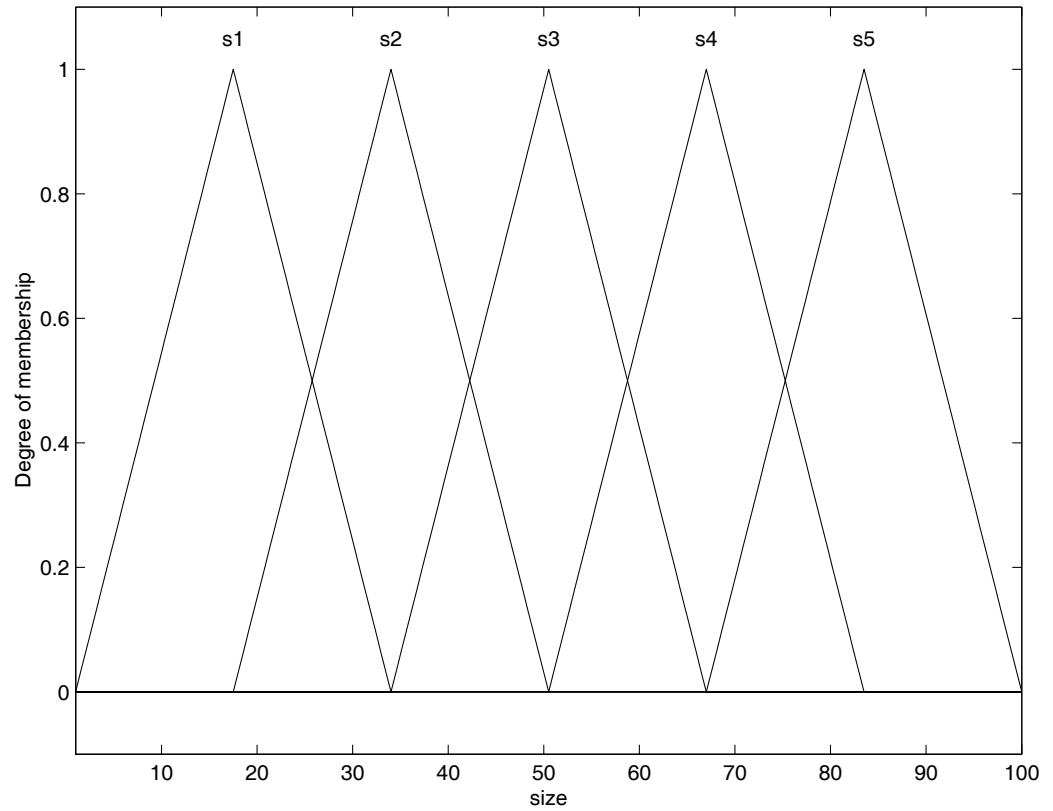
#### **6.2 Experimental Design for Nominal Effort Prediction**

The diagram of Figure 23 gives a summary description of the two different approaches we have chosen for the experiments. In the first approach (Approach I), we modify the output fuzzy set of a firing rule, and modify all the fuzzy sets of the antecedent. In the second approach, we modify the output fuzzy set of a firing rule and modify only one fuzzy set of the antecedent – the fuzzy set that yields the minimum membership degree for the current input pattern of all the fuzzy sets in the antecedent of the rule under consideration. We have 8 different experiments for each approach, as given by the leaf nodes of each route, for a total of 16 experiments.



**Figure 23: The different Experimental Routes for Nominal Effort Training**

In order to partition the input/output variables domain into fuzzy regions represented by fuzzy sets, as discussed in Algorithm 1.1, we have investigated two alternative ways to define our membership functions - shouldered and unshouldered membership functions (MFs). The shouldered MFs means that the distribution of the MFs is uniform within the variable domain, but the first and the last MFs are shouldered, as shown in Figure 20. The unshouldered MFs uniformly partition the input space into fuzzy regions of same fuzzy sets sizes. An example of unshouldered MFs is given in Figure 24, which represents an alternative partitioning of the same *size* variable of Figure 20. The shouldered membership functions have been discovered to be more meaningful as they cover the entire input space as against the unshouldered MFs that do not cover the initial and the end parts of the input space appropriately. Experiments have also revealed the superiority of shouldered MFs in terms of prediction accuracy as will be shown later in this chapter.



**Figure 24: Divisions of the input variable SIZE into fuzzy regions and the corresponding membership functions using Unshouldered MFs**

Each of the two partitioning methods has two different approaches for calculating the output error during training – normalized and non-normalized error calculation methods. To justify the effect of normalizing the output error, we carried out different experiments involving normalization of the output error, and others not involving normalization of output error for each of the two approaches considered.



The leaf nodes give two terminating conditions. In the first option, training stops after a specified number of epochs (max epoch), and the current fuzzy inference system (FIS) at the end of the training becomes the trained FIS for estimation. In case of the second option, training continues for a specified number of epochs, as in the first case, but we continually save the FIS and the epoch at which we obtained the minimum root-mean-square relative error (RMSRE). At the end of our training, the saved FIS with minimum RMSRE is taken as the trained FIS for estimation.

According to our experiments to be discussed in Section 6.4, the path in bold represents the most promising track of all approaches we have proposed.

### **6.3 Cost Drivers Implementation**

Our cost driver experiments followed from the discussion in Chapter 4, where the cost drivers were categorized for fuzzy logic implementation. Of the 15 cost drivers, 11 of them are directly amenable to fuzzification. The other four, shown in the grey area of Table 12 and Table 13 were not studied because of the reasons given earlier.

We created 11 fuzzy inference systems (FIS), each FIS implementing the relationship between each cost driver and its effects on predicted nominal effort. The membership functions definition for the cost drivers in the

antecedent of each rule are guided by the definitions of the cost drivers ratings given in Table 12. The effects of the cost driver on nominal effort in the consequent are modeled using membership functions derived from the knowledge in the effort multipliers of Table 13.

The defuzzified result from each of the FIS developed for the cost drivers are crisp values specifying the effect the cost driver has on nominal effort. The results are aggregated in a similar manner to the COCOMO algorithmic model, to adjust the nominal effort.

While we have not carried out extensive validation of our developed FIS for the cost drivers because of non-availability of data, the transparency in the development of the rulebase of each FIS and our approach to membership functions definitions show that the fuzzy inference system will give multipliers to the same accuracy level as the COCOMO model. Besides, our inference systems have additional advantage of handling the jump between different ratings of the cost drivers as discussed during the sensitivity analysis of Chapter 3. The FIS realizing the cost drivers also permit the incorporation of expert knowledge in defining the fuzzy sets.

The membership functions and rules formulated for each FIS modeling different cost drivers are presented in Appendix B. For clarity, we show a table of the cost driver definitions and our transformation of the knowledge

into membership functions of the antecedents of the rules (see Figure 55 for antecedent of the DATA CD). The same is done for the effort multipliers in the consequent (see Figure 56 for consequent of the DATA CD). The remaining CDs are given in Figure 57 through Figure 76 in Appendix B.

## 6.4 Experimental Results and Discussion

This section presents our experiments investigating the feasibility of the different tracks shown in Figure 23 for nominal effort prediction. The various experiments cover extensively all the tracks we have identified earlier.

The dataset used for the experiments investigating the approaches in this work are artificial datasets randomly generated following the procedure discussed in Chapter 5. The original dataset generated, and the partitioned training and validation dataset are contained in Appendix A. All *sizes* used for our experiments in thousands of delivered source instructions fall within 1 – 100KDSI.

The objectives of the five experiments in this section are summarized in Table 15:

**Table 15: Summary of the Experimental Objectives**

EXPERIMENT	OBJECTIVE
Experiment #I	Investigating the feasibility of different tracks of Approach I (Adapting all antecedent MFs)
Experiment #II	Investigating the feasibility of different tracks of Approach II (Adapting antecedent MF yielding min. degree of membership)
Experiment #III	Comparing the Accuracy and Reasonability of Experiment #I and Experiment #II for a decision on the best track
Experiment #IV	Validating the trained FIS using COCOMO database
Experiment #V	Investigating the reasonability of the Training Algorithms

Before discussing the experiments, we first present the measure used in evaluating the prediction quality of our developed fuzzy inference systems.

#### **6.4.1 Evaluating Prediction Accuracy**

In order to enable us compare the prediction accuracy of our training procedure to the actual values, we need a quantitative measure of prediction quality. The quantities we have used in these experiments are RMSRE and prediction at level  $q$  –  $PRED(q)$ .

Suppose we have a set of  $n$  projects, let  $k$  be the number of them whose mean magnitude of relative error is less than or equal to  $q$ . Then, we have:

$$PRED(q) = k/n \quad \text{Equation 16}$$

Conte *et al.* [12] suggest that an acceptable level for mean magnitude of relative error is something less than or equal to 0.25. This notion was used to define a measure of prediction quality. For example, if  $PRED(0.25) = 0.47$ , then 47% of the predicted values fall within 25 % of their actual values. It is desirable to always maximize this value and minimize RMSRE.

#### **6.4.2 Experiment I – Adapting all Input Membership Functions**

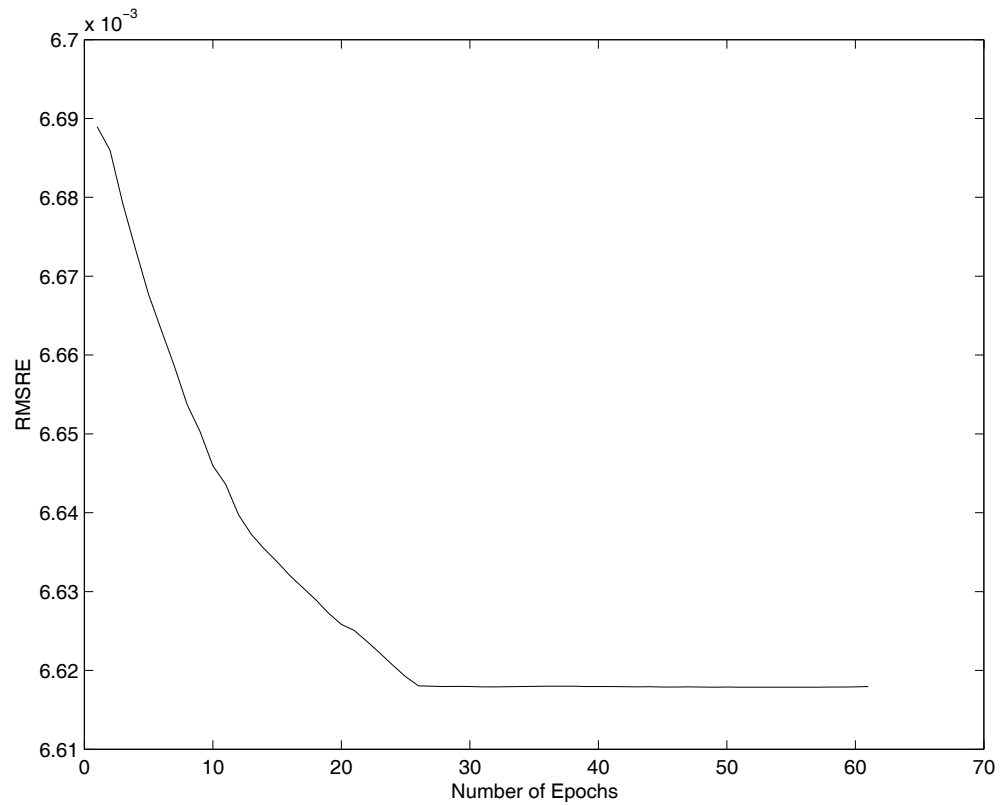
This experiment investigates the suitability of Approach I in the experimental design diagram of Figure 23. This approach, as discussed earlier, adapts the consequent membership function and adapts the membership functions of all the fuzzy sets used in all antecedents of contributing rules. We present our experimental results based on the two different routes that are applicable using this approach – shouldered and unshouldered membership functions.

##### **6.4.2.1 Approach I - Using Shouldered Fuzzy Sets**

The shouldered MFs option introduced in the experimental design diagram uses two alternative approaches to calculate the error in output of the fuzzy system – normalized error and non-normalized error.

The first run of our experiment uses 5 input MFs for the *size* input variable and 3 input MFs for *mode* input variable (this applies to all our experiments). A fuzzy inference system was built, based on this specification, and then

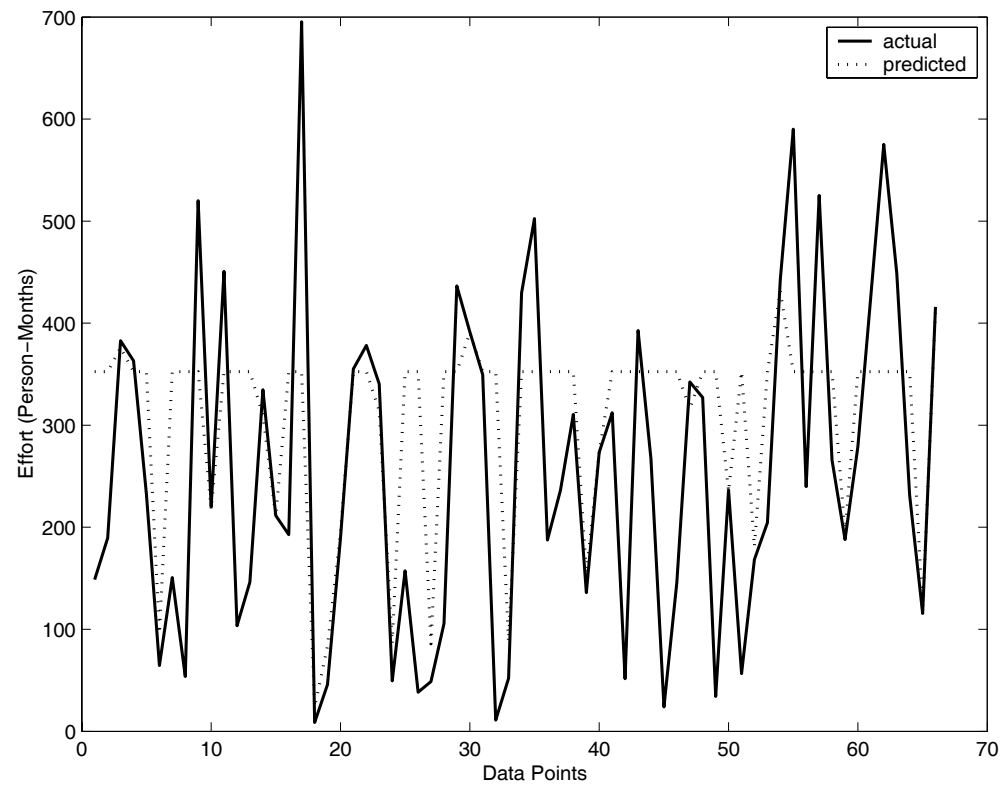
trained. The prediction of the trained fuzzy inference system (FIS) with the normalized and non-normalized error measures is carried out on the actual data on which the FIS was trained and on the validation data. A plot of the RMSRE during training of the shouldered-normalized error track with 5 MFs is shown in Figure 25. The error graph shows the learning curve of the training algorithm when applied to the current approach.



**Figure 25: The Error graph during Training of the shouldered-normalized error FIS of Approach I**

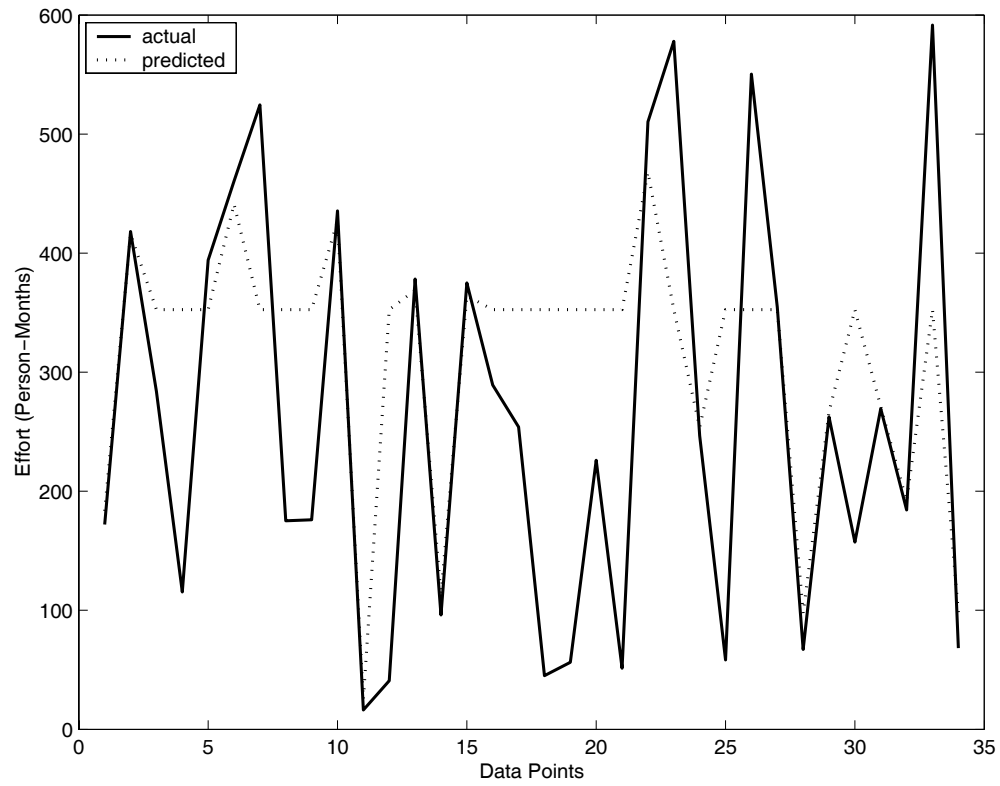
Once the training is complete, we must test the trained FIS with a set of test examples to see how well it performs. To help evaluate the performance of our trained FIS using this method, we have used the training data and validation data for testing.

The prediction of the trained FIS and the true values on training dataset is given in Figure 26. Figure 27 also shows similar prediction result using the validation dataset that was not used during training. Table 16 shows the detailed results of a repeat of the same experiment, using 3, 5, 7, 9, and 11 MFs respectively for the *size* input variable. The table gives detailed testing results, including error measures (RMSRE) that gives a reflection of the discrepancy in the estimates made by the trained and untrained FISs, using both training and validation datasets for testing. From Figure 26 and Figure 27, we can see that the shouldered MF track of Approach I does not show good approximation, even after training the FIS. The resulting poor performance is more of the approach adapting all the antecedents MFs, which does not localize the effect of the errors in output to those MFs that directly determine the contribution of each rule to the current output.



**Figure 26: Prediction of effort using the Training dataset for testing of shouldered-normalized error FIS of Approach I**





**Figure 27: Prediction of effort using the Validation dataset for testing of shouldered-normalized error FIS of Approach I**

Similar experiments to those described above have been carried out for the shouldered non-normalized error track of Approach I; the summary table of the results is also given in Table 17. The results in Table 16 and Table 17 show that Approach I generally performs poorly after training than untrained FIS using prediction quality, PRED(25).

The Tables show seven major column headings with the following meaning:

1. Column 1 gives the number of MFs used for the *size* input variable in each experiment (the number of MFs for *mode* is always 3)
2. Column 2 gives the prediction quality of the trained (After) and untrained (Before) FIS, when testing with the dataset used for training. Prediction quality is always tested after training.
3. Column 3 gives the RMSRE of the trained (After) and untrained (Before) FIS while testing with the dataset used for training.
4. Column 4 represents the same meaning as in (2.) above, but testing the FISs with the validation data that was never used during training
5. Column 5 represents the same meaning as in (3.) above, but gives RMSRE of the FISs while testing with the validation data that was never used during training
6. Column 6 shows the minimum RMSRE of each experiment during training. The RMSRE is used to evaluate the performance of the FIS during training, and it guides the modifications during training. The FIS saved with minimum RMSRE is the FIS taken as the best-trained FIS. The minimum RMSRE can be achieved during any of the training epochs or at the final epoch.

7. Column 7 shows the number of training epochs for each experiment

Each row entry in the tables represents one experiment with the specified number of *size* input MFs. This same interpretation applies to all the tables of our experiments realizing the tracks of Figure 23.

Considering the shouldered MFs experiments reported in Table 16 and Table 17, it can be seen that the prediction quality of trained/untrained FISs increases with increasing number of MFs. Our intuitive explanation for this is related to the suitability of a rulebase, which depends on the initial fuzzy partitions. That is, if there are too few fuzzy sets, groups of data that should be represented by different rules might be covered by a single rule. In the same vein, if we have more fuzzy sets than necessary, too many rules would be created resulting in over-fitting. Over-fitting may prevent the trained FIS from generalizing, or producing correct outputs when presented with data that was not used in training. The accuracy of the trained FIS increases, but interpretability of the rulebase decreases. On the other hand, the RMSRE decreases with increasing number of MFs. The explanation again follows from the gain in prediction quality.

The increase in values of RMSRE after training when compared to lower values recorded for untrained FIS, as given in Table 16 and Table 17 might

intuitively be the result of modifying MFs that are not necessarily active during the propagation of an input pattern. This undesired modification would not focus on the rules contributing to the current output, thereby distorting the meaning of the rulebase, and resulting in lower prediction quality and higher RMSRE.

In addition, results shown in Table 16 and Table 17 further confirm that the increase in RMSRE after training is a result of the modification of MFs and not because of normalization/non-normalization of output errors. The two tables show the consistency in this increment and consistency in reduction of prediction quality after training.

**Table 16: Summary of Prediction Quality using Shouldered and Normalized Error for five Trained FIS with different number of fuzzy sets for the SIZE input variable, showing PRED(25) on training and testing datasets for Approach I.**

No. MF	Prediction Accuracy Pred(25) on Training Data (%)		RMSRE on Training Data		Prediction Accuracy Pred(25) on Validation Data (%)		RMSRE on Validation Data		Min. RMSRE <sup>7</sup> During Training	No. of Training Epochs
	Before Training	After Training	Before Training	After Training	Before Training	After Training	Before Training	After Training		
3	57.58	42.42	2.0852	4.7227	55.88	44.12	1.8628	2.4996	0.006718	21
5	77.27	43.94	1.0315	4.6523	76.47	47.06	0.8637	2.4399	0.006618	53
7	84.85	45.45	0.6761	4.6465	82.35	52.94	0.5080	2.4448	0.006610	87
9	92.42	50.00	0.5009	4.6487	91.18	47.06	0.3367	2.4405	0.006613	36
11	95.45	51.52	0.3960	4.6463	94.12	52.94	0.2450	2.4374	0.006609	54

---

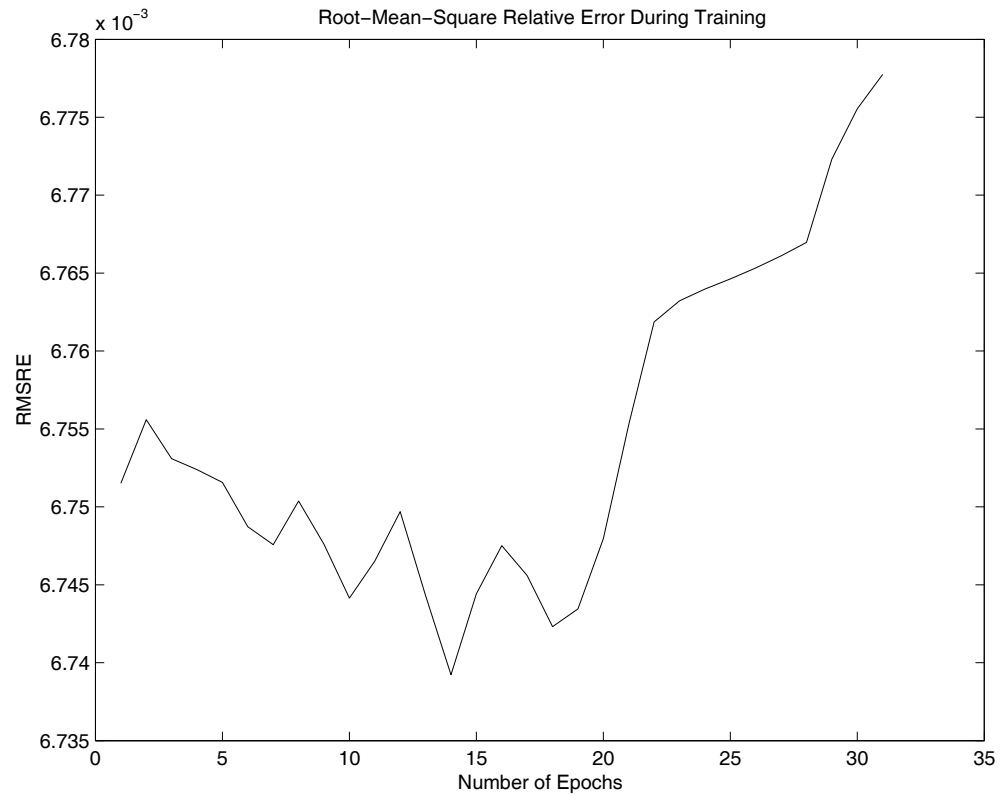
<sup>7</sup> Please note that the values of RMSRE in column 6 and column 3 (after) are supposed to be almost same, since they operate on the same training data. The *normalization* of output error during training caused column 6 to be considerably lower. If we normalize errors during testing, we will have column 3 (after) give same result as column 6. The normalization is not necessarily done during testing, since we require the RMSRE to compare trained/untrained.

**Table 17: Summary of Prediction Quality using Shouldered and Non-normalized Error for five Trained FIS using different number of fuzzy sets for the SIZE input variable, showing PRED(25) on training and testing datasets for Approach I.**

No. MF	Prediction Accuracy PRED (25) on Training Data (%)		RMSRE on Training Data		Prediction Accuracy PRED(25) on Validation Data (%)		RMSRE on Validation Data		Min. RMSRE During Training	No. of Training Epochs
	Before Training	After Training	Before Training	After Training	Before Training	After Training	Before Training	After Training		
3	57.58	30.30	2.0852	6.9101	55.88	26.47	1.8628	4.4481	6.8971	1
5	77.27	46.97	1.0315	4.6949	76.47	52.94	0.8637	2.4747	4.6689	3
7	84.85	51.52	0.6761	4.6540	82.35	52.94	0.5080	2.4451	4.6536	6
9	92.42	51.52	0.5009	4.6496	91.18	52.94	0.3367	2.4415	4.6493	9
11	95.45	51.52	0.3960	4.6452	94.12	52.94	0.2450	2.4384	4.6454	9

#### 6.4.2.2 Approach I - Using Un-Shouldered Fuzzy Sets

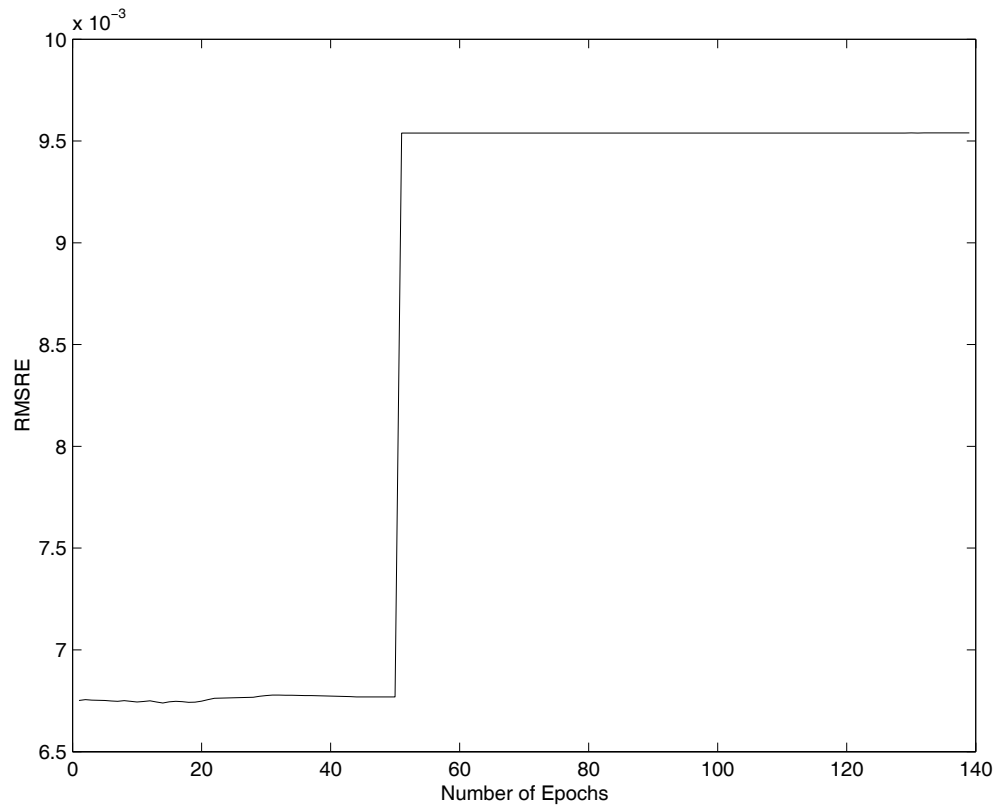
Experiments on the un-shouldered MFs track of Approach I follow from our discussion of the shouldered MFs. We have investigated both the normalized error and non-normalized error tracks of un-shouldered MFs here. The error graph during training for the un-shouldered normalized error track with 5 input MFs for *size*, is given in Figure 28.



**Figure 28: The Error graph during Training of the Un-shouldered-normalized error FIS of Approach I**

From the RMSRE graph, using un-shouldered approach, training errors does not converge to a desirable solution, and we only have saved intermediate FIS with minimum root mean square relative error (RMSRE) during training. The RMSRE during training will start-off decreasing, but it will eventually start increasing and never comes down, even when trained for more epochs. Figure 29 also shows the same experiment, when carried out for 139 training epochs. Meanwhile, training dataset error is naturally expected to decrease with number of epochs of training.



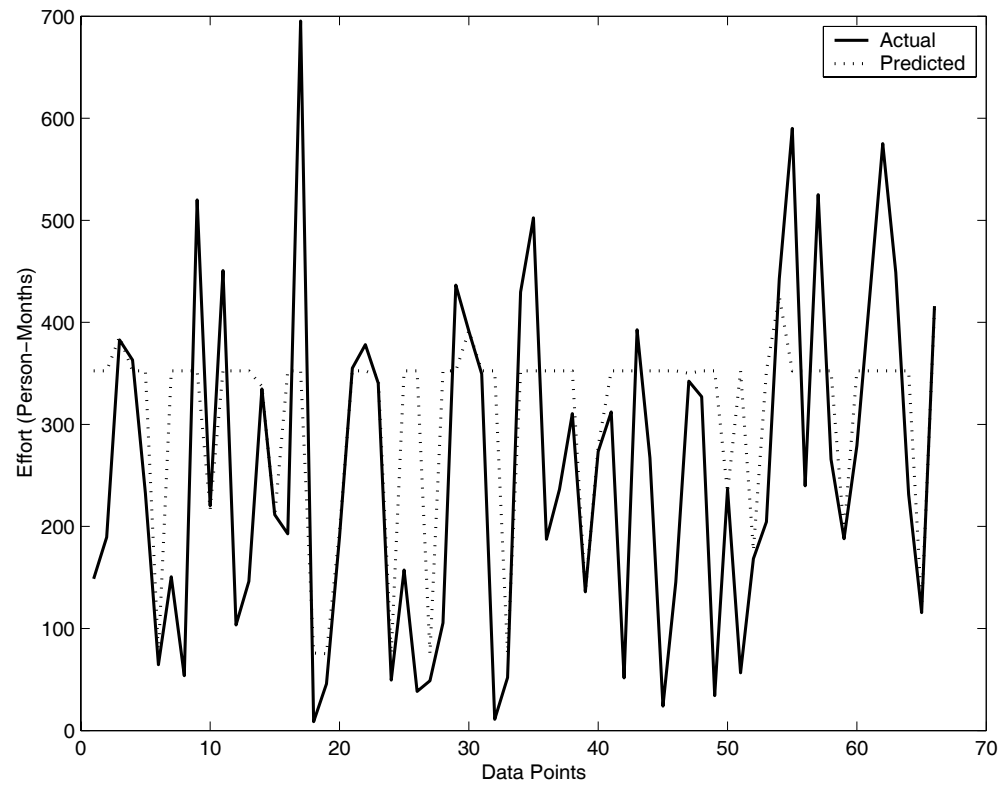


**Figure 29: The Error graph during Training of the Un-shouldered-normalized error FIS of Approach I after 139 Epochs**

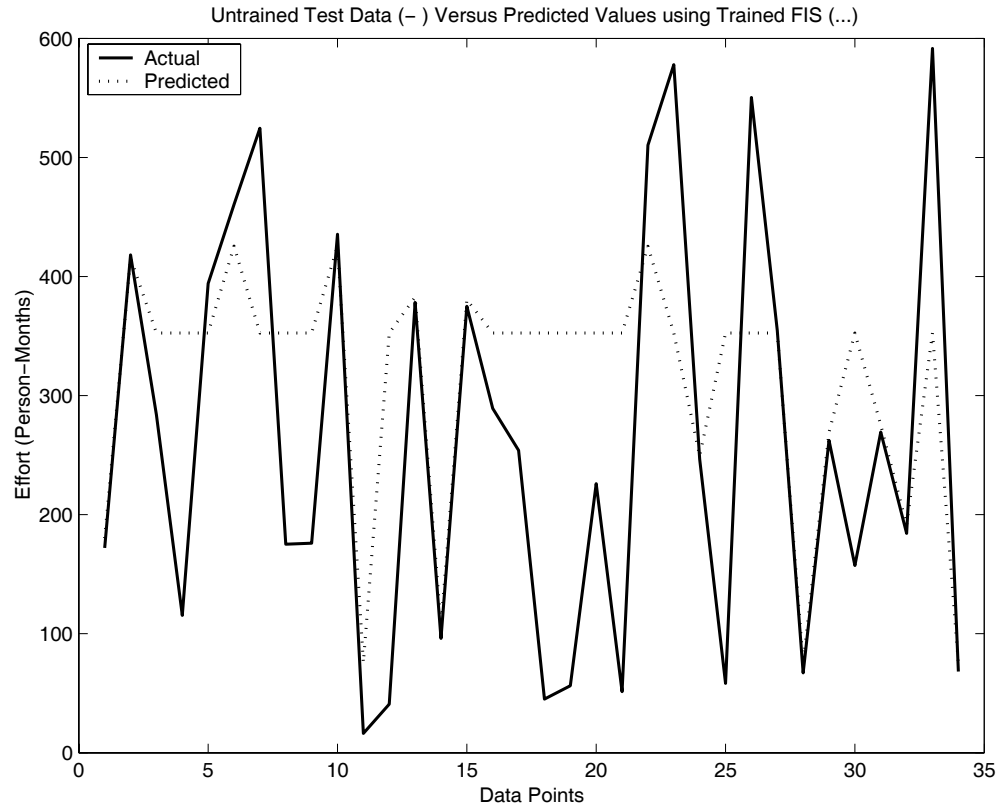
Using our intuition, the increase in RMSRE can be explained in relation to over-fitting of the FIS to those points that are well covered by the MFs used in partitioning the input space. Data points from the training dataset that belongs to the points not well covered, for example, the initial and final part of the input region would consistently result in large errors. This implies that the trained FIS would not be able to generalize over the entire input space.

The un-shouldered MF is therefore not an encouraging route to explore, coupled with the fact that the shouldered MF is more meaningful as explained earlier.

Figure 30 and Figure 31 present testing results using trained FIS on training and validation datasets for the un-shouldered approach, in the same way explained for shouldered approach.



**Figure 30: Prediction of effort using the Training dataset for validation of Un-shouldered-normalized error FIS of Approach I**



**Figure 31: Prediction of effort using the Testing dataset for validation of Un-shouldered-normalized error FIS of Approach I**

The summaries presented in Table 18 and Table 19 contain results from repeat experiments with 3, 5, 7, 9, and 11 fuzzy sets for the size input variable using normalized and non-normalized error measures. The results

shown in the tables after training are not encouraging as the trained FIS performs poorly than untrained FIS. This is the same case for the shouldered MF of the same Approach I.

**Table 18: Summary of Prediction Quality using Un-shouldered and Normalized Error for five Trained FIS using different number of fuzzy sets for the SIZE input variable, showing PRED(25) on training and testing datasets for Approach I.**

No. MF	Prediction Accuracy PRED (25) on Training Data (%)		RMSRE on Training Data		Prediction Accuracy PRED(25) on Validation Data (%)		RMSRE on Validation Data		Min. RMSRE During Training	No. of Training Epochs
	Before Training	After Training	Before Training	After Training	Before Training	After Training	Before Training	After Training		
3	75.76	43.94	2.0693	4.8991	73.53	44.12	1.3684	2.6599	0.006969	13
5	84.85	45.45	1.2439	4.7405	82.35	52.94	0.7425	2.5150	0.006739	14
7	92.42	51.51	0.8598	4.6867	91.18	52.94	0.46664	2.4706	0.006660	21
9	95.45	51.51	0.6377	4.6687	94.12	52.94	0.3212	2.4529	0.006641	20
11	95.45	51.51	0.4977	4.6617	97.06	52.94	0.2345	2.4461	0.006627	1

**Table 19: Summary of Prediction Quality using Un-shouldered and Non-normalized Error for five Trained FIS using different number of fuzzy sets for the SIZE input variable, showing PRED(25) on training and testing datasets for Approach I.**

No. MF	Prediction Accuracy PRED (25) on Training Data (%)		RMSRE on Training Data		Prediction Accuracy PRED(25) on Validation Data (%)		RMSRE on Validation Data		Min. RMSRE During Training	No. of Training Epochs
	Before Training	After Training	Before Training	After Training	Before Training	After Training	Before Training	After Training		
3	75.76	30.30	2.0693	6.9101	73.53	26.47	1.3684	4.4481	4.757	2
5	84.85	51.52	1.2439	6.7041	82.35	52.94	0.7425	2.4593	4.674	14
7	92.42	50.00	0.8598	4.6688	91.18	47.06	0.4664	2.4529	4.6692	14
9	95.45	51.52	0.6377	4.6636	94.12	52.94	0.3212	2.4481	4.6636	3
11	95.45	51.52	0.4977	4.6521	97.06	52.94	0.2345	2.4393	4.6521	10

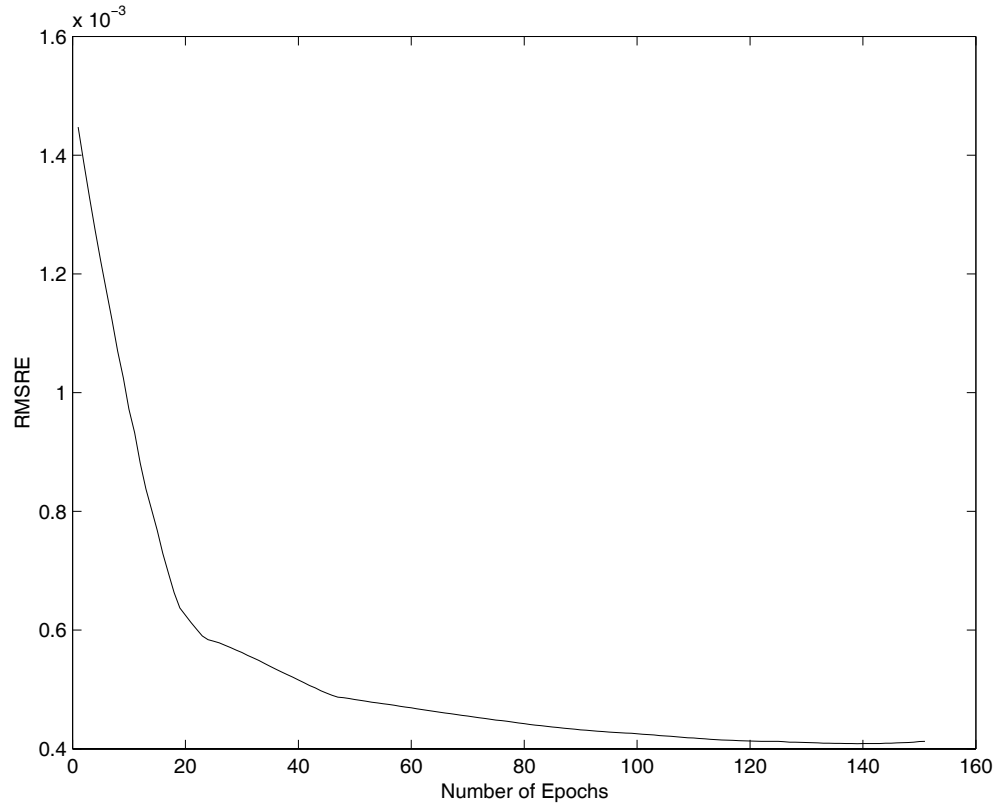
### **6.4.3 Experiment II - Adapting Input MFs yielding Minimum Membership Degree**

The purpose of the experiments presented here is to investigate the suitability of Approach II in the experimental design diagram of Figure 23. This approach adapts the consequent MFs and adapts the MF of only one fuzzy set in the antecedent that yields minimum degree of membership for the current input pattern during training.

Meanwhile, the experimental procedures here are same as those discussed in experiment I of Section 6.4.2 investigating Approach I. The shouldered and un-shouldered fuzzy sets approaches are both investigated.

#### **6.4.3.1 Approach II - Using Shouldered Fuzzy Sets**

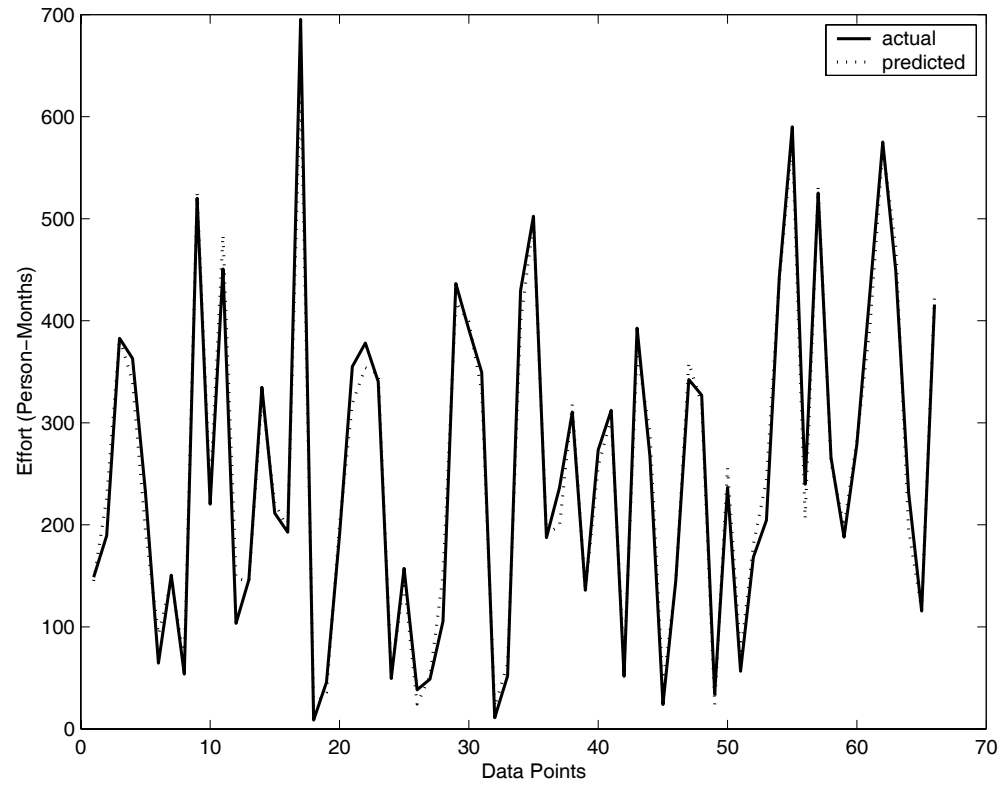
The first run of our experiment uses 5 input MFs for the *size* input variable and 3 for *mode* input variable, as discussed for Approach I (A sample FIS with this MFs configuration and complete rulebase is given in Appendix C). A fuzzy inference system was built, based on this specification, and then trained. Testing the prediction quality of the trained FIS with normalized and non-normalized error measures was carried out on the training dataset used for training the FIS and on the validation dataset. The error (RMSRE) graph during training of the shouldered-normalized error track using 5 MFs for *size* is given in Figure 32. The RMSRE reduces as training progresses until it converges to a reasonable minimum.



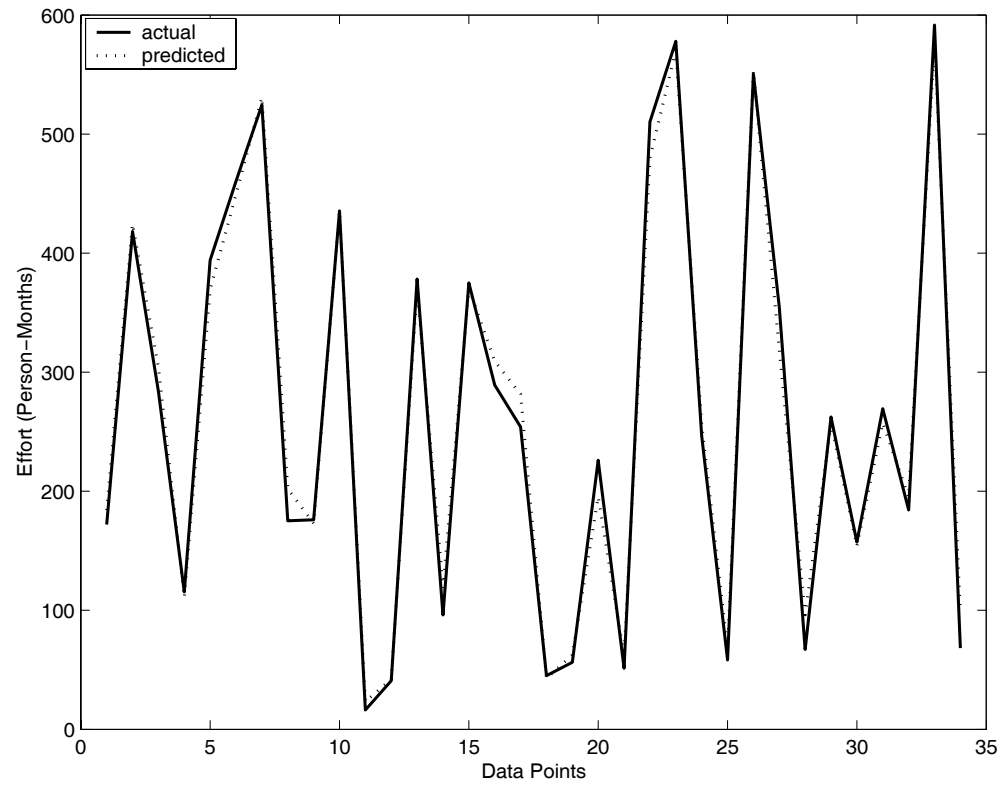
**Figure 32: The Error graph during Training of the shouldered-normalized error FIS of Approach II**

Figure 33 shows a testing of the effort prediction capabilities of the trained FIS on the actual dataset used for training, while Figure 34 shows similar prediction using the validation dataset that was not used during training.





**Figure 33: Prediction of effort using the Training dataset for testing of shouldered-normalized error FIS of Approach II**



**Figure 34: Prediction of effort using the Testing dataset for validation of shouldered-normalized error FIS of Approach II**

Table 20 shows the detailed results of a repeat of the same experiments, using 3, 5, 7, 9, and 11 fuzzy sets respectively. The table gives detailed validation results including the error measures during training and validation for each experiment as discussed earlier for Approach I.

Similar experiments to those described above have been carried out for the shouldered non-normalized error track; the summary table of the results is

also given in Table 21. The results in Table 20 and Table 21 show this approach to be very promising with the prediction quality of trained FIS matching or out-performing that of untrained FIS, and giving lower error (RMSRE) during validation testing. These promising results were not achieved by Approach I.

**Table 20: Summary of Prediction Quality using Shouldered, Normalized Error for five Trained FIS using different number of fuzzy sets for the SIZE input variable, showing PRED(25) on training and testing datasets for Approach II .**

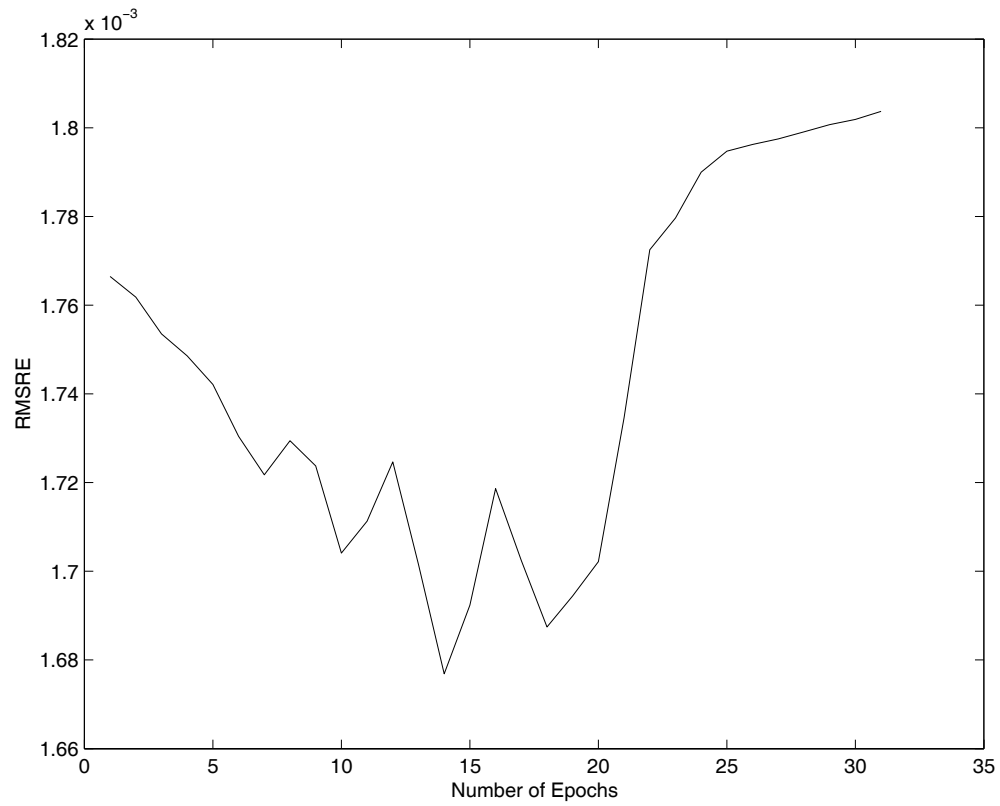
No. MF	Prediction Accuracy PRED (25) on Training Data (%)		RMSRE on Training Data		Prediction Accuracy PRED(25) on Validation Data (%)		RMSRE on Validation Data		Min. RMSRE During Training	No. of Training Epochs
	Before Training	After Training	Before Training	After Training	Before Training	After Training	Before Training	After Training		
3	57.58	74.24	2.0852	1.0063	55.88	73.53	1.8628	0.71054	0.0014283	20
5	77.27	86.36	1.0315	0.2875	76.47	91.18	0.8637	0.1535	0.0004087	139
7	84.85	90.91	0.6761	0.2061	82.35	88.24	0.5080	0.2006	0.0002931	209
9	92.42	95.45	0.5008	0.2549	91.18	91.18	0.3367	0.1735	0.0003798	151
11	95.45	95.45	0.3960	0.2313	94.12	94.12	0.2450	0.1239	0.0003292	122

**Table 21: Summary of Prediction Quality using Shouldered, Non-normalized Error for five Trained FIS using different number of fuzzy sets for the SIZE input variable, showing PRED(25) on training and testing datasets for Approach II.**

No. MF	Prediction Accuracy PRED (25) on Training Data (%)		RMSRE on Training Data		Prediction Accuracy PRED(25) on Validation Data (%)		RMSRE on Validation Data		Min. RMSRE During Training	No. of Training Epochs
	Before Training	After Training	Before Training	After Training	Before Training	After Training	Before Training	After Training		
3	57.58	40.91	2.0852	2.9040	55.88	26.47	1.8628	2.1800	2.30080	9
5	77.27	74.24	1.0315	0.7494	76.47	79.41	0.8637	0.4556	0.85286	1
7	84.85	87.88	0.6761	0.4327	82.35	82.35	0.5080	0.2988	0.40510	6
9	92.42	95.45	0.5009	0.3769	91.18	97.06	0.3367	0.2322	0.31058	12
11	95.45	95.45	0.3960	0.1413	94.12	88.24	0.2450	0.1451	0.1125	29

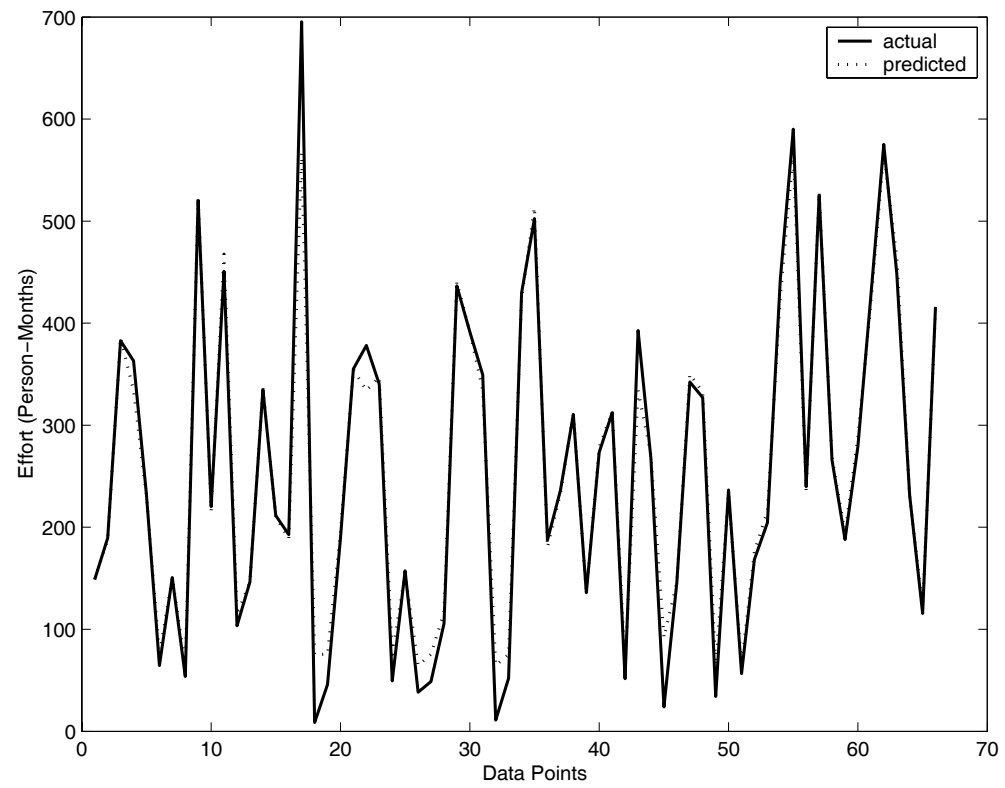
#### **6.4.3.2 Approach II – Using Un-Shouldered Fuzzy Sets**

Experiments on the un-shouldered membership functions track of approach II follow from our discussion of the shouldered fuzzy sets track of approach II. We have investigated both the normalized error and non-normalized error tracks as well. The error graph during training for the un-shouldered normalized error track with 5 input membership functions, for example, is given in Figure 35. The error graph during training is similar to that of un-shouldered track of Approach I. The interpretation of the nature of the graph follows from the discussion for Approach I.



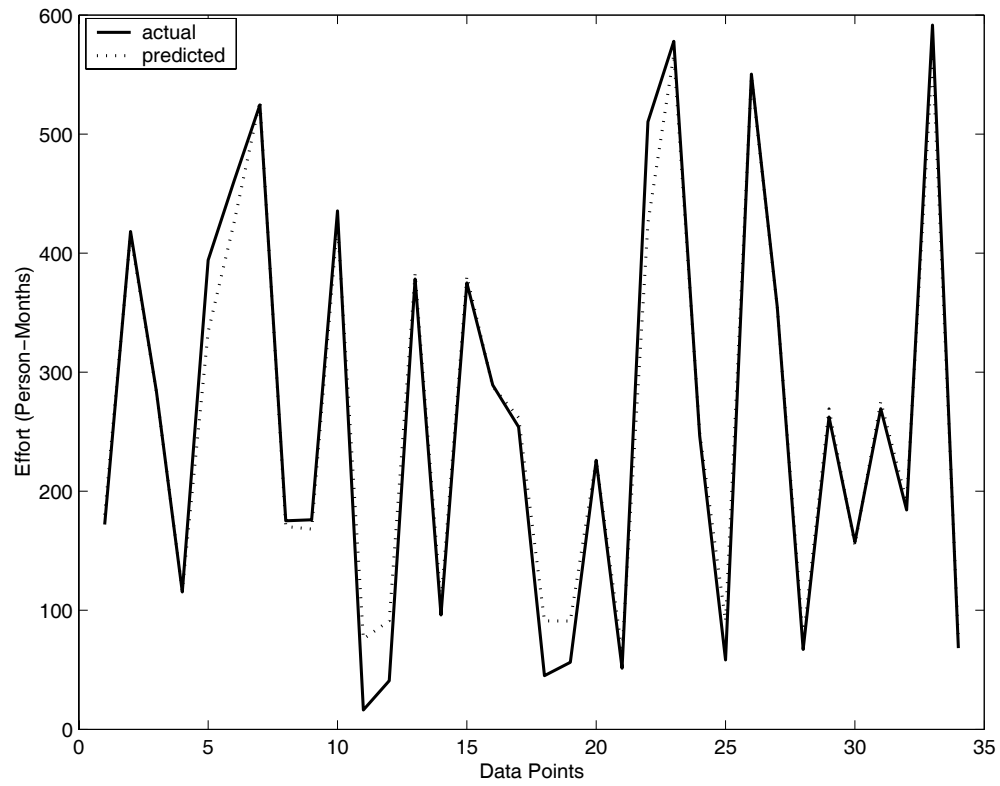
**Figure 35: The Error graph during training of the un-shouldered-normalized error FIS of Approach II**

The results obtained from the un-shouldered track are encouraging, as given in Figure 36 and Figure 37 during validation testing of one experiment, the non-convergence of the error during training remains a problem that places the credibility of the approach under question.



**Figure 36: Prediction of effort using the Training dataset for validation of Un-shouldered-normalized error FIS of Approach II**





**Figure 37: Prediction of effort using the Testing dataset for validation of Un-shouldered-normalized error FIS of Approach II**

The summaries presented in Table 22 and Table 23 contain results from repeat experiments with 3, 5, 7, 9, and 11 fuzzy sets for the size input variable using normalized and non-normalized error measures respectively. The results shown in the tables after training for all the runs are equally encouraging as the trained FIS clearly matches or out-performs untrained FIS.

**Table 22: Summary of Prediction Quality using Un-shouldered, Normalized Error for five Trained FIS using different number of fuzzy sets for the SIZE input variable, showing PRED(25) on training and testing datasets for Approach II.**

No. MF	Prediction Accuracy PRED (25) on Training Data (%)		RMSRE on Training Data		Prediction Accuracy PRED(25) on Validation Data (%)		RMSRE on Validation Data		Min. RMSRE During Training	No. of Training Epochs
	Before Training	After Training	Before Training	After Training	Before Training	After Training	Before Training	After Training		
3	75.76	75.76	2.0693	4.9919	73.53	73.53	1.3684	1.2822	0.0028025	12
5	84.85	86.36	1.2439	1.1857	82.35	82.35	0.7425	0.7033	0.0016769	14
7	92.42	92.42	0.8598	0.8052	91.18	91.18	0.4664	0.4411	0.0011410	21
9	95.45	95.45	0.6377	0.5995	94.12	94.12	0.3212	0.3011	0.0008500	20
11	95.45	95.45	0.4977	0.4716	97.06	97.06	0.2345	0.2196	0.0006696	24

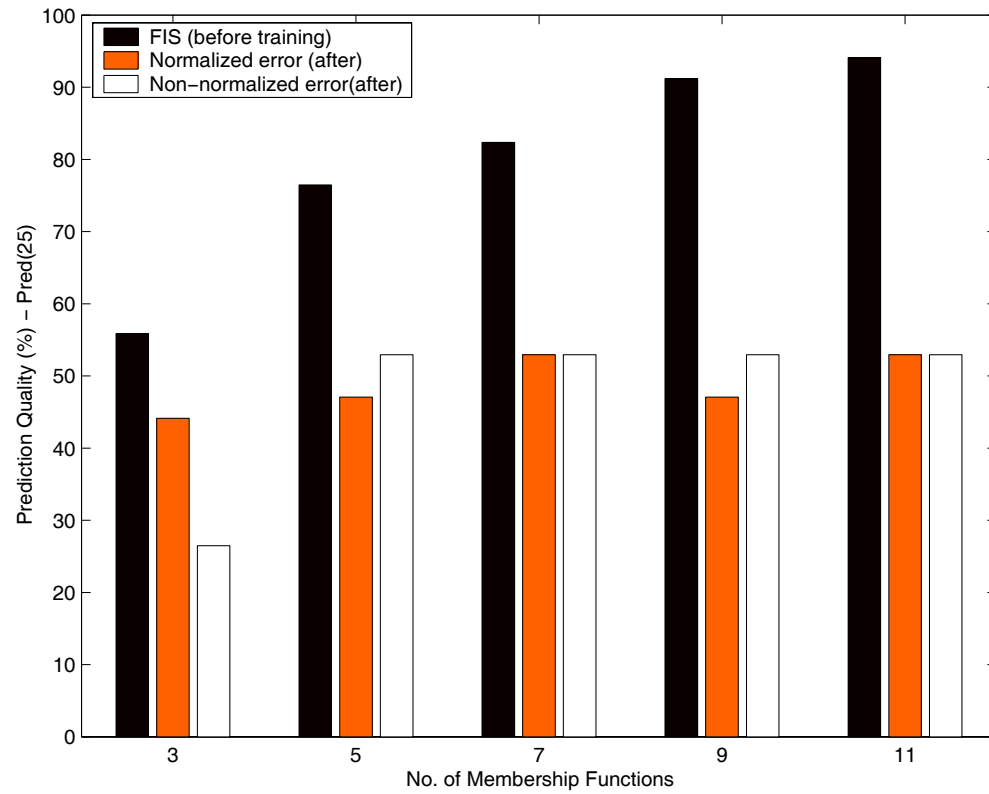
**Table 23: Summary of Prediction Quality using Un-shouldered, Non-normalized Error for five Trained FIS using different number of fuzzy sets for the SIZE input variable, showing PRED(25) on training and testing datasets for Approach II.**

No. MF	Prediction Accuracy PRED (25) on Training Data (%)		RMSRE on Training Data		Prediction Accuracy PRED(25) on Validation Data (%)		RMSRE on Validation Data		Min. RMSRE During Training	No. of Training Epochs
	Before Training	After Training	Before Training	After Training	Before Training	After Training	Before Training	After Training		
3	75.76	40.91	2.0693	2.2501	73.53	29.41	1.3684	1.7357	1.3019	5
5	84.85	86.36	1.2439	4.8524	82.35	76.47	0.7425	0.3634	0.6750	5
7	92.42	93.94	0.8598	4.8512	91.18	97.06	0.4664	0.3306	0.6701	7
9	95.45	95.45	0.6377	0.1193	94.12	85.29	0.3212	0.1531	0.1955	31
11	95.45	95.45	0.4977	0.1305	97.06	88.24	0.2345	0.1380	0.1310	26

#### **6.4.4 Experiment III - Comparing Prediction Accuracy of the two Approaches**

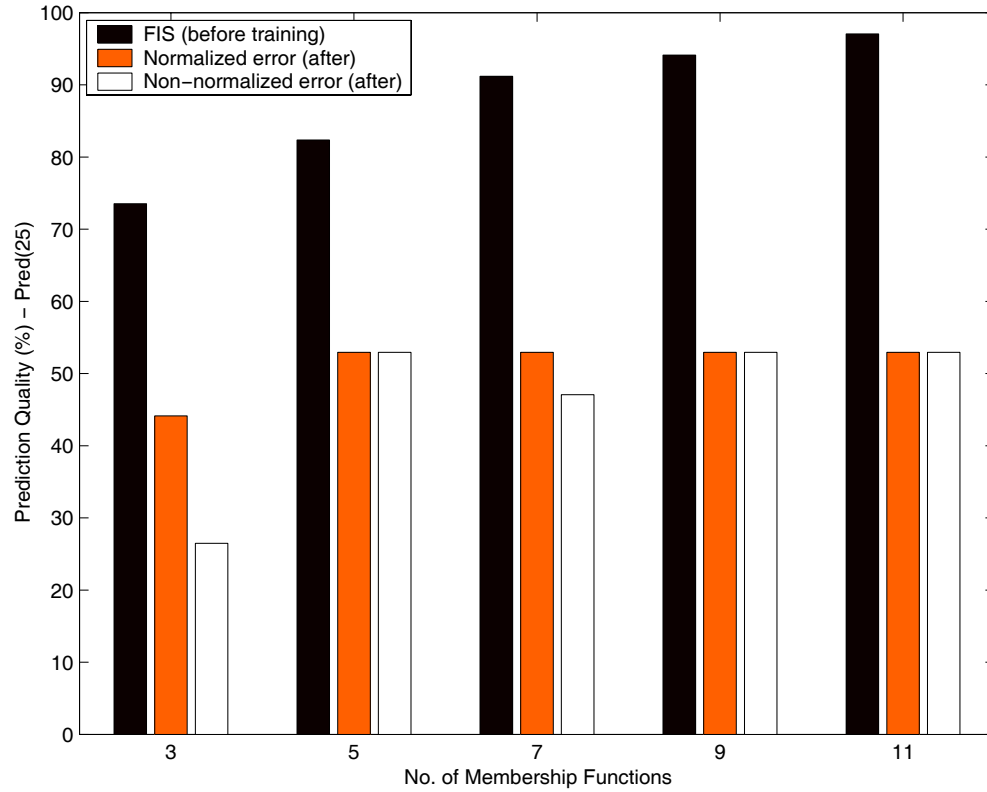
Having presented our experimental results obtained from the different approaches we proposed, the question remains: which one is most suitable for effort prediction. This section evaluates and compares the different routes earlier introduced to answer this question. Evaluation is based on experimental results summaries shown in Table 16 - Table 23. The testing results reported for validation dataset in the tables have been used in this section in order to test the generalization ability of the trained FIS, since the validation dataset was not seen during training.

The chart in Figure 38 represents the prediction quality of the shouldered membership function track of Approach I. The chart is derived from the prediction quality,  $PRED(25)$ , on validation data given in Table 16 and Table 17. It shows prediction quality before training and after training for normalized and non-normalized error measures tracks, using different number of MFs for the input variable *size*.



**Figure 38: The prediction quality of shouldered MFs using normalized and non-normalized error measures of Approach I on validation dataset before and after training.**

Figure 39, however, represents the prediction quality of the un-shouldered membership function track of Approach I. The chart is derived from the prediction quality, PRED(25), on validation data given in Table 18 and Table 19. It presents a clearer picture of the prediction quality before training and after training for normalized and non-normalized error measures, using un-shouldered membership functions.

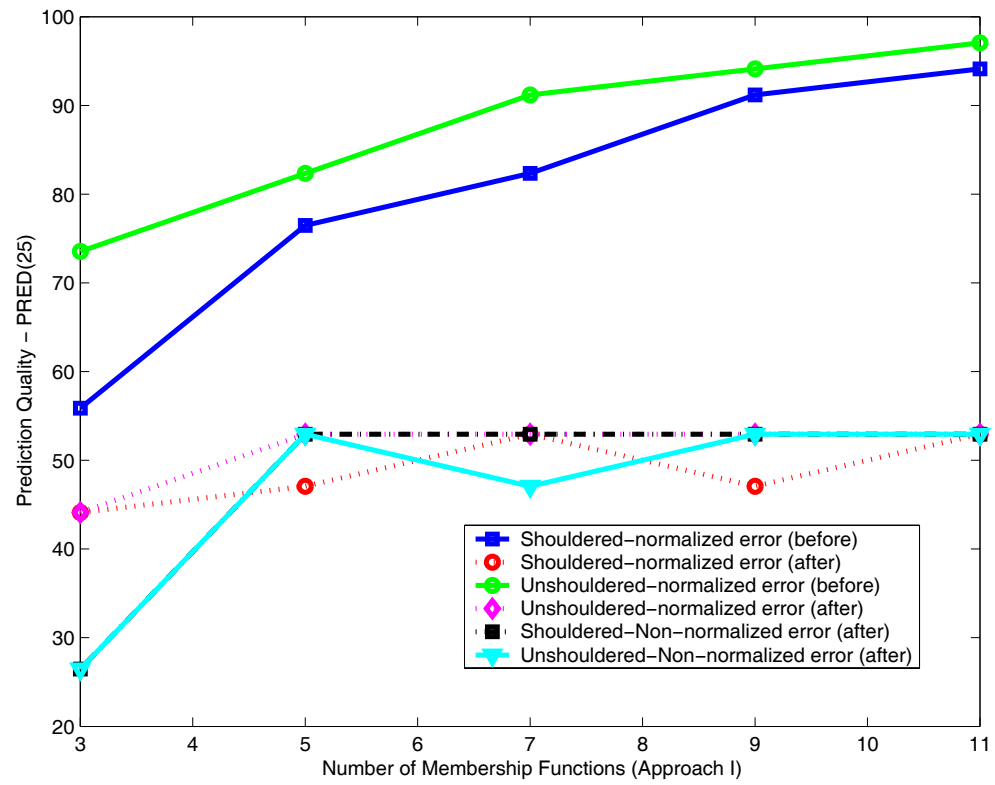


**Figure 39: The prediction quality of Un-shouldered MFs using normalized and non-normalized error measure of Approach I on validation dataset before and after training.**

A look at the two charts (i.e. Figure 38 and Figure 39) above reveals that Approach I is not promising as it consistently gives worse performance after training. In all cases, the prediction quality reduces and the RMSRE increases after training. Figure 40 shows the low prediction quality of all the tracks of Approach I after training as compared to untrained FIS in clearer

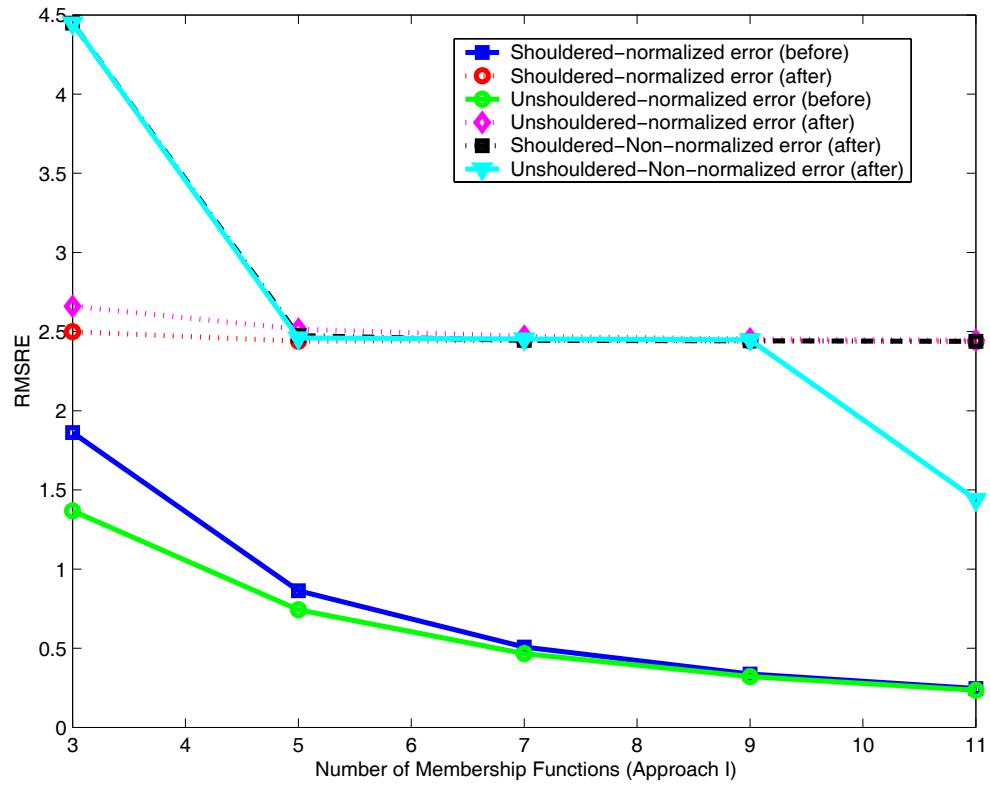
picture. On the other hand, Figure 41 shows the RMSRE to be higher for trained FIS than untrained FIS. This is against our objective of achieving increased prediction quality and low RMSRE.

The reason for this apparent poor performance may be that the modifications done to the MFs are not being focused on the MFs that determine the contribution of each rule as stated earlier. Rather, the effect of the fuzzy system error is distributed across the whole antecedents MFs, even those that do not support the current input pattern, which makes the modification less sensible. In real sense, if any training approach cannot optimize the performance of the rulebase of a fuzzy inference system, the performance should at least not reduce. This leads us to the evaluation of Approach II.



**Figure 40: Graph of Prediction Quality of different tracks of Approach I using the validation dataset**



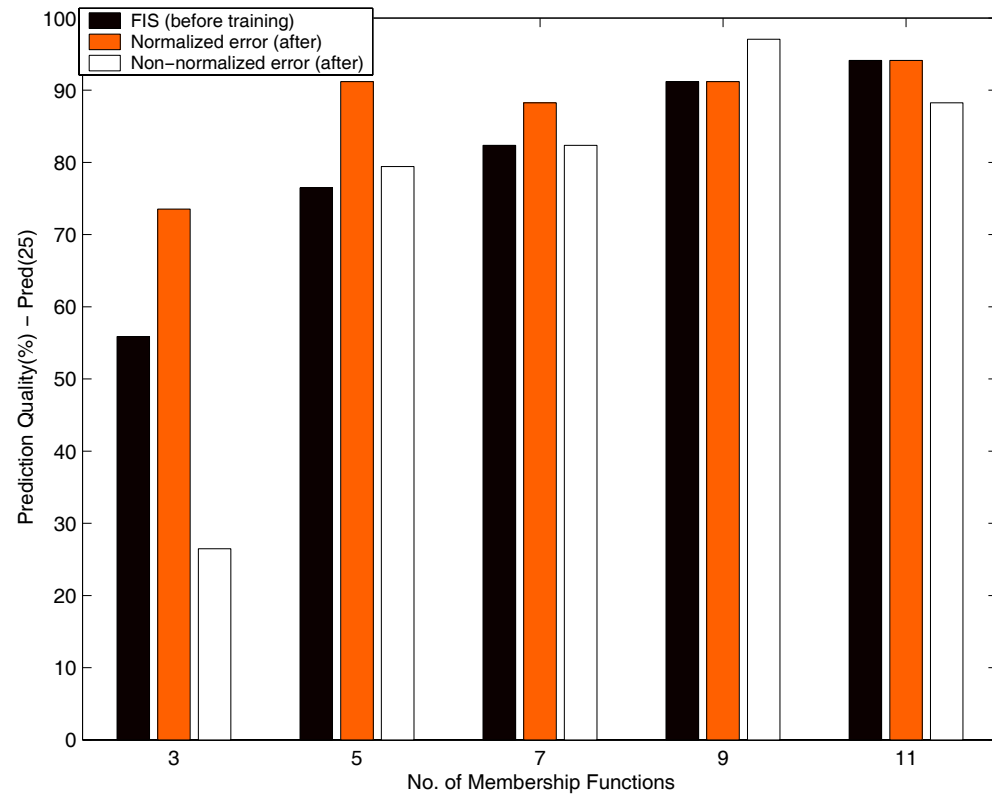


**Figure 41: Graph of RMSRE of different tracks of Approach I using the Validation dataset**

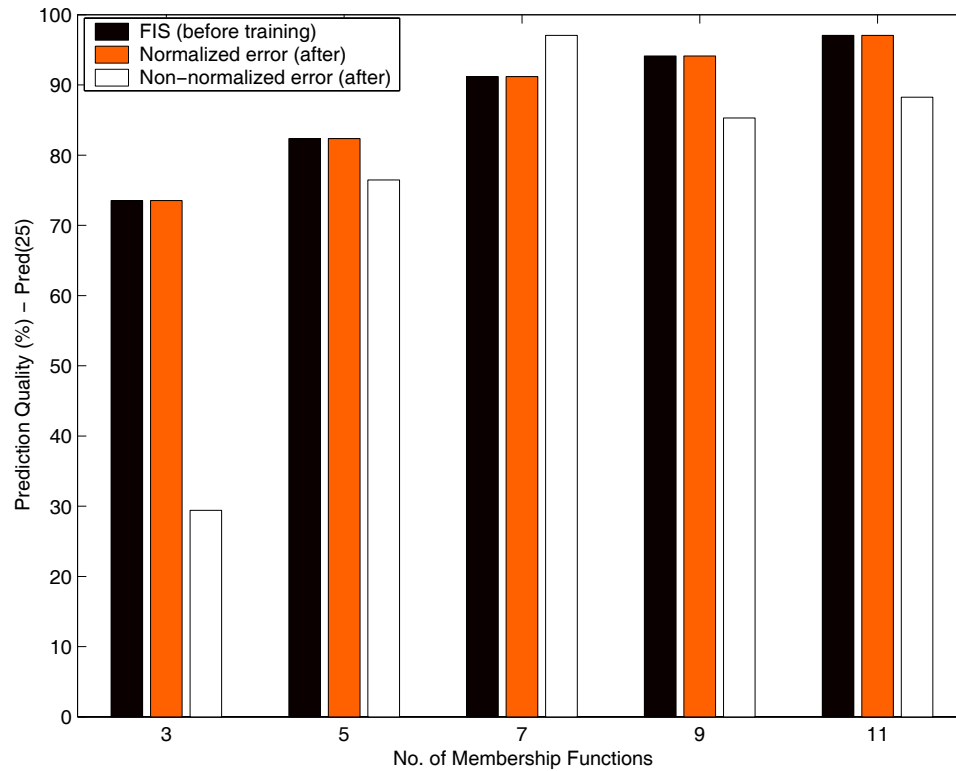
In evaluating Approach II, we first showed in charts, the improvement in prediction quality achieved when compared to untrained FIS. We then proceed to compare the shouldered and un-shouldered tracks of the approach in order to arrive at a decision on the best track.

The chart in Figure 42 is derived from the prediction quality on validation data of Table 20 and Table 21, while the chart of Figure 43 is derived from

the prediction quality on validation data of Table 22 and Table 23, as discussed for Approach I.



**Figure 42: The prediction quality of shouldered MFs using normalized and non-normalized error measure of Approach II on validation dataset before and after training.**



**Figure 43: The prediction quality of Un-shouldered MFs using normalized and non-normalized error measure of Approach II on validation dataset before and after training.**

From Figure 42 specifically, there are considerable improvements in the prediction quality of the trained FIS when compared to untrained FIS. The prediction quality also increases with increasing number of input MFs. This consistency is particularly noted for normalized error of the shouldered MF track. The non-normalized error track, while it shows some improvements, does not produce such consistency. The prediction quality decreases for some selected number of input MFs with non-normalized error, specifically

for 3 and 11 MFs, we observe drops as compared to untrained FIS (see Figure 43).

However, the un-shouldered MFs track of Approach II presented in Figure 43 only show one single improvement for the non-normalized error track. The normalized error track of un-shouldered MFs did not improve over the untrained FIS, but there is no drop in the prediction quality as well. The non-improvement in prediction quality for un-shouldered MFs might be because of the method of partitioning the input MFs, which does not equally cover the entire domain. The high prediction quality reported for untrained FIS may be a result of the ability to make good approximation in the regions that are well covered, but not at the two extremes that are not well covered.

The discussions of the two approaches above have focused primarily on the increase/decrease in prediction quality after training. In order to compare the shouldered and un-shouldered tracks of Approach II to make informed decision, we again consider the results of predictions made using the validation datasets. Table 24 is extracted from the prediction quality on validation dataset of Table 20 - Table 23, while Table 25 is extracted from the RMSRE on validation dataset of Table 20 - Table 23. These two tables (i.e., Table 24 and Table 25) allow us to compare shouldered and un-shouldered MFs track of Approach II.

**Table 24: Comparing Prediction Quality of Shouldered/Un-Shouldered MFs tracks using the Validation dataset for normalized and non-normalized errors.**

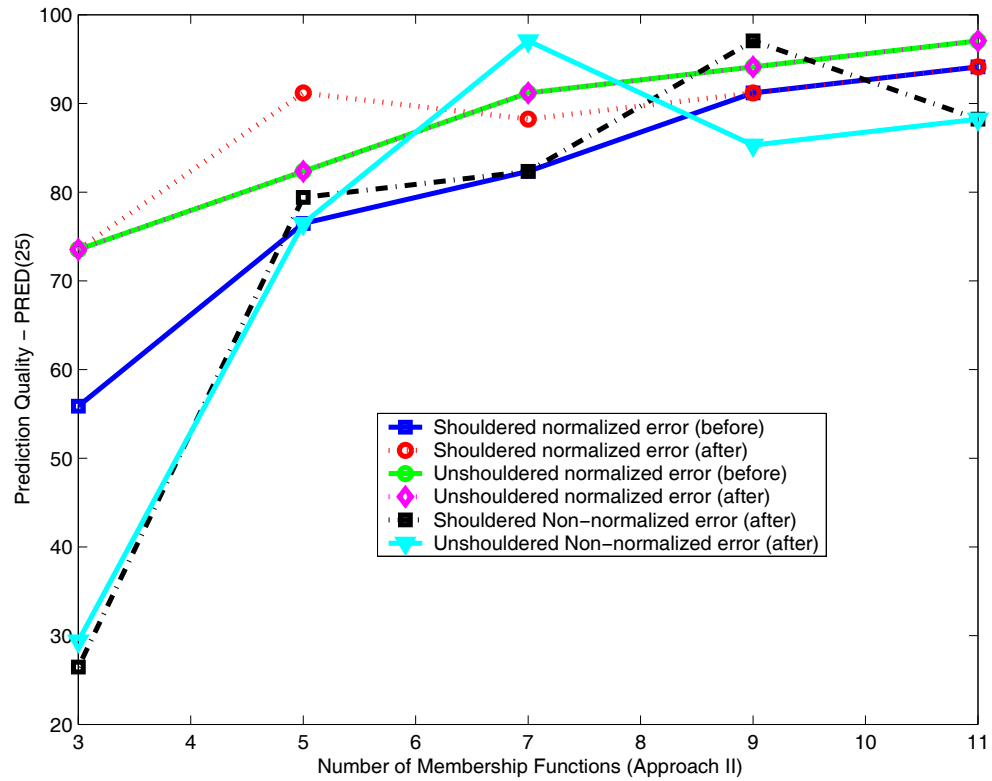
Number of MFs	Normalized Error Using Validation Dataset				Non-normalized Error Using Validation Dataset			
	Shouldered MF PRED(25)		Un-shouldered MF PRED(25)		Shouldered MF PRED(25)		Un-shouldered MF PRED(25)	
	Before Training	After Training	Before Training	After Training	Before Training	After Training	Before Training	After Training
3	55.88	73.53	73.53	73.53	55.88	26.47	73.53	29.41
5	76.47	91.18	82.35	82.35	76.47	79.41	82.35	76.47
7	82.35	88.24	91.18	91.18	82.35	82.35	91.18	97.06
9	91.18	91.18	94.12	94.12	91.18	97.06	94.12	85.29
11	94.12	94.12	97.06	97.06	94.12	88.24	97.06	88.24

**Table 25: Comparing RMSRE of Shouldered/Un-Shouldered MFs tracks using the Validation dataset for normalized and non-normalized errors.**

Number of MFs	Normalized Error Using Validation Dataset				Non-normalized Error Using Validation Dataset			
	Shouldered MFs RMSRE		Un-shouldered MFs RMSRE		Shouldered MFs RMSRE		Un-shouldered MFs RMSRE	
	Before Training	After Training	Before Training	After Training	Before Training	After Training	Before Training	After Training
3	1.8628	0.7105	1.3684	1.2822	1.8628	2.1800	1.3684	1.7357
5	0.8637	0.1535	0.7425	0.7033	0.8637	0.4556	0.7425	0.3634
7	0.5080	0.2006	0.4664	0.4411	0.5080	0.2988	0.4664	0.3306
9	0.3367	0.1735	0.3212	0.3011	0.3367	0.2322	0.3212	0.1531
11	0.2450	0.1239	0.2345	0.2196	0.2450	0.1451	0.2345	0.1380

From Table 24, we can compare the performance of the shouldered and un-shouldered tracks using their prediction qualities. The graph shown in Figure 44 gives a summary representation that makes the comparison easier.

In Figure 44 below, we have not explicitly drawn the non-normalized error “*before*” for both shouldered and un-shouldered MFs. The reason is that, whether we train with normalized or non-normalized error, the same FIS is used. Therefore, the “*before*” FIS used for normalized error training is the same “*before*” FIS used for non-normalized training. Please refer to Table 25 to see clearly that the “*before*” columns of shouldered MF is the same for both normalized and non-normalized errors, and the same thing happens for the un-shouldered tracks.



**Figure 44: Graph of Prediction Quality of Shouldered/Un-Shouldered MFs tracks using the Validation dataset for normalized and non-normalized errors.**

From the graph of Figure 44 the only track that showed consistent improvement in prediction quality always, is the shouldered-normalized track, as observed earlier. The shouldered non-normalized showed some improvements at some points, precisely, when we have 5 and 9 MFs respectively, but the reduction in quality when 3 MFs are used results in inconsistencies.



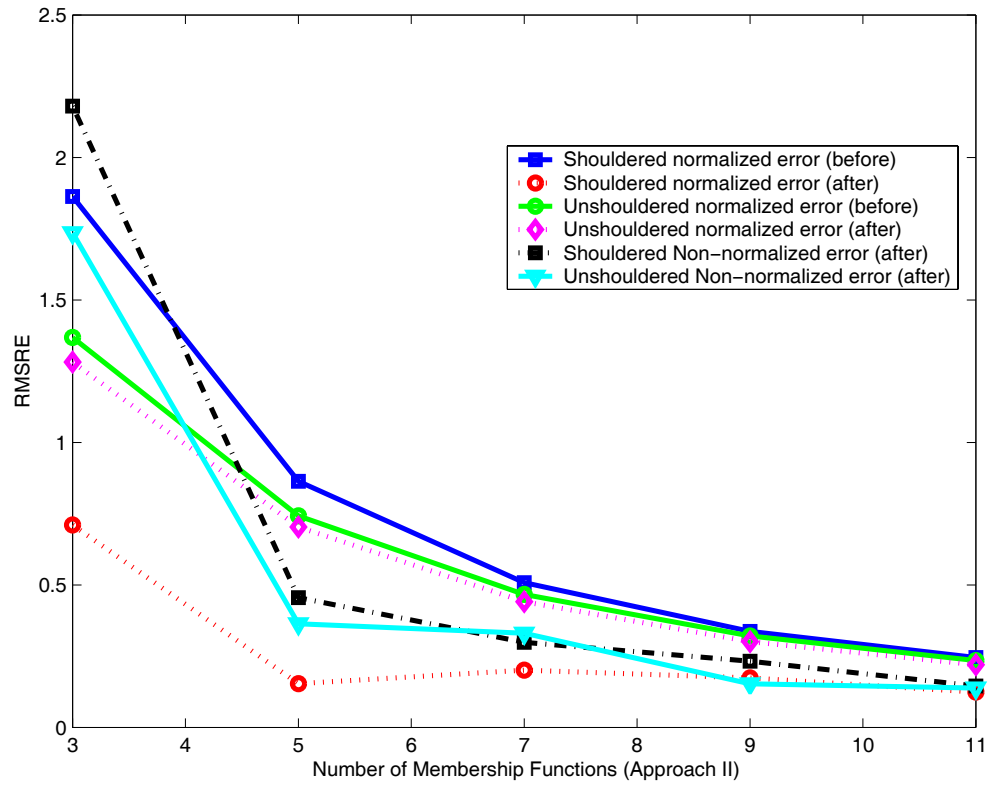
For the un-shouldered MF track, the prediction quality is not different for trained and untrained FIS using normalized error. The non-normalized error option of un-shouldered MF track does not maintain a steady increase/decrease it rather oscillates. The reason for this behavior of non-normalized error may be due to the over-emphasized modifications to the MFs for large errors in the output of the fuzzy system as discussed in Chapter 5.

Making a decision to choose between shouldered-normalized and un-shouldered-normalized is not straightforward because the un-shouldered-normalized has higher prediction quality when 7, 9 and 11 MFs were used than the shouldered-normalized. Therefore, we need additional information to aid our judgment. This information is revealed in the RMSRE graph shown in Figure 45. The reduction in the error values (RMSRE) for shouldered-normalized track after training is drastic and very significant when compared to all other tracks. However, this result is not obtained for un-shouldered-normalized. In sort, the closest in consistency as regards RMSRE reduction, is the shouldered, non-normalized track.

The consistent improvement in prediction quality and consistent reduction in RMSRE positively points to shouldered-normalized as the most favorable track. The reduction in RMSRE again implies that there are many data points

that are very close to the 25% crisp cut chosen for measuring quality (PRED(25)). This crisp cut has shadowed the visibility of the improvement recorded after training by hiding those data points that are very close to the 25% error margin, but the RMSRE revealed this improvement in Figure 45.

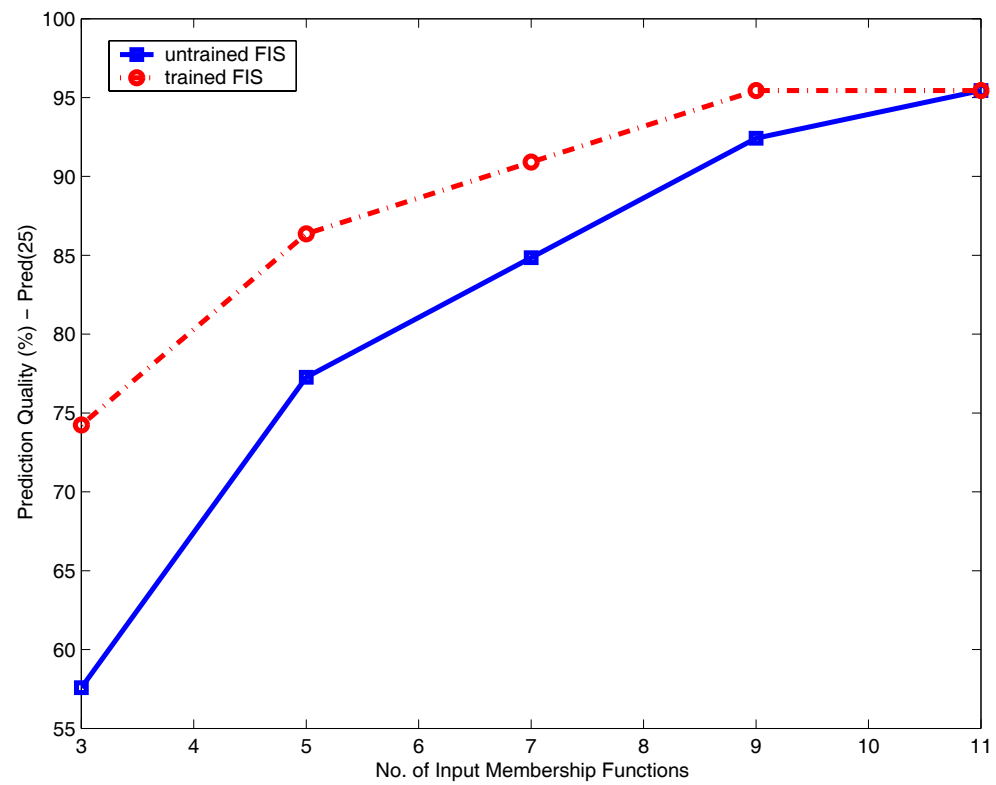
The reason given above for Figure 44, where we did not explicitly redraw the non-normalized error graph “*before*” for both shouldered and un-shouldered MFs tracks, also applies to Figure 45.



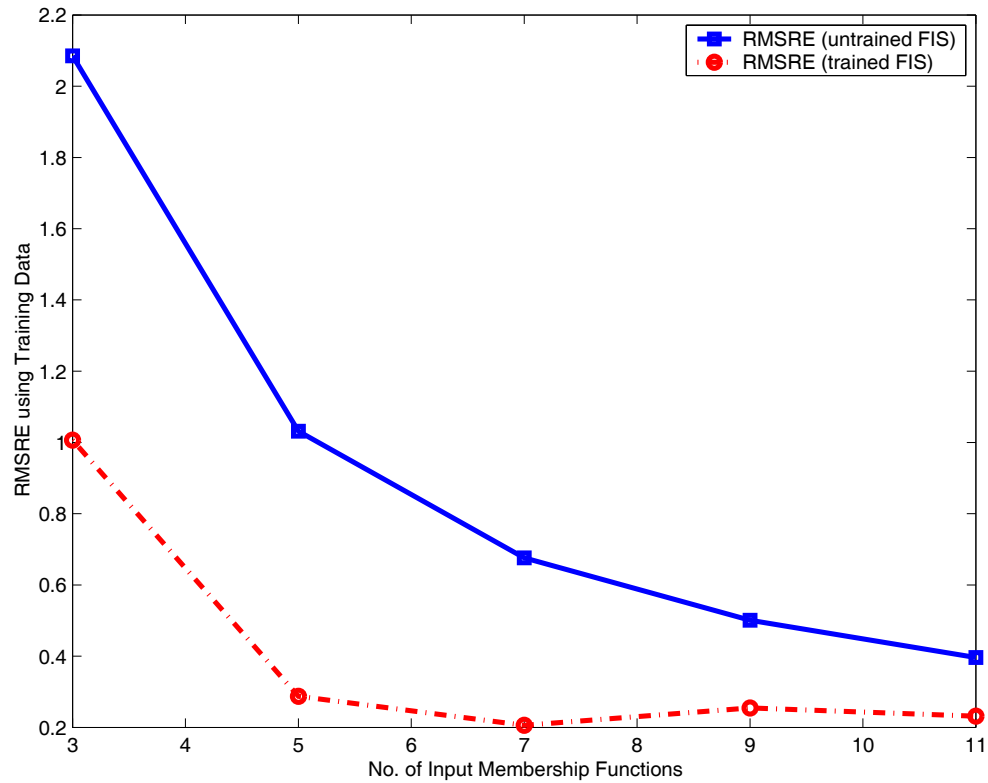
**Figure 45: Graph of RMSRE of Shouldered/Un-Shouldered MFs tracks using the Validation dataset for normalized and non-normalized errors.**

Considering our discussion so far on comparing the prediction capabilities of the various tracks, it is pragmatic to conclude that the *shouldered normalized error* track of Approach II is the best of all the 16 tracks explored. Beside the high prediction quality, the root mean square relative error (RMSRE) during prediction is considerably and consistently lower than that of untrained FIS.

To conclude this discussion on our choice of *shouldered-normalized error* track, we plot graphs of its PRED(25) and RMSRE using the training dataset to show the consistency irrespective of the dataset used for testing. Figure 46 is a graph showing the improvement in the performance of the trained shouldered-normalized error track with increasing number of input membership functions, using results on the training data in Table 20 in lieu of the validation data we have been using in this experiment. Figure 47 shows corresponding reduction in the RMSRE with increasing number of input MFs using training data for testing. The importance and meaningfulness of this low RMSRE would be further discussed when evaluating the effectiveness of our training algorithm.



**Figure 46: The prediction quality of Untrained and Trained FIS using different Number of MFs.**



**Figure 47: The RMSRE of the Untrained and Trained FIS using different Number of MFs.**

#### 6.4.5 Experiment IV – Validation Using the COCOMO Database

The validation of our approaches to building trained FIS for effort prediction has been done using artificial datasets in the previous section. While real life data has always been adjudged difficult to obtain in software engineering community, the public availability of the COCOMO database offers some respite in this regard.

For the validation in this section, we have selected 53 of the 63 live projects in the COCOMO database [3] whose sizes in lines of code fall within 1-100 KDSI. This selected dataset is contained in Appendix A.

We adopted two approaches to our validation using the COCOMO database. The first validation approach uses the FIS already trained with artificial training dataset as done in the previous experiments. This validation procedure is presented in Section 6.4.5.1. Secondly, we trained a new FIS using the data from COCOMO database directly, and testing the performance on the COCOMO data after training as presented in Section 6.4.5.2.

#### **6.4.5.1 Validation using Datasets from COCOMO Database on Trained FIS**

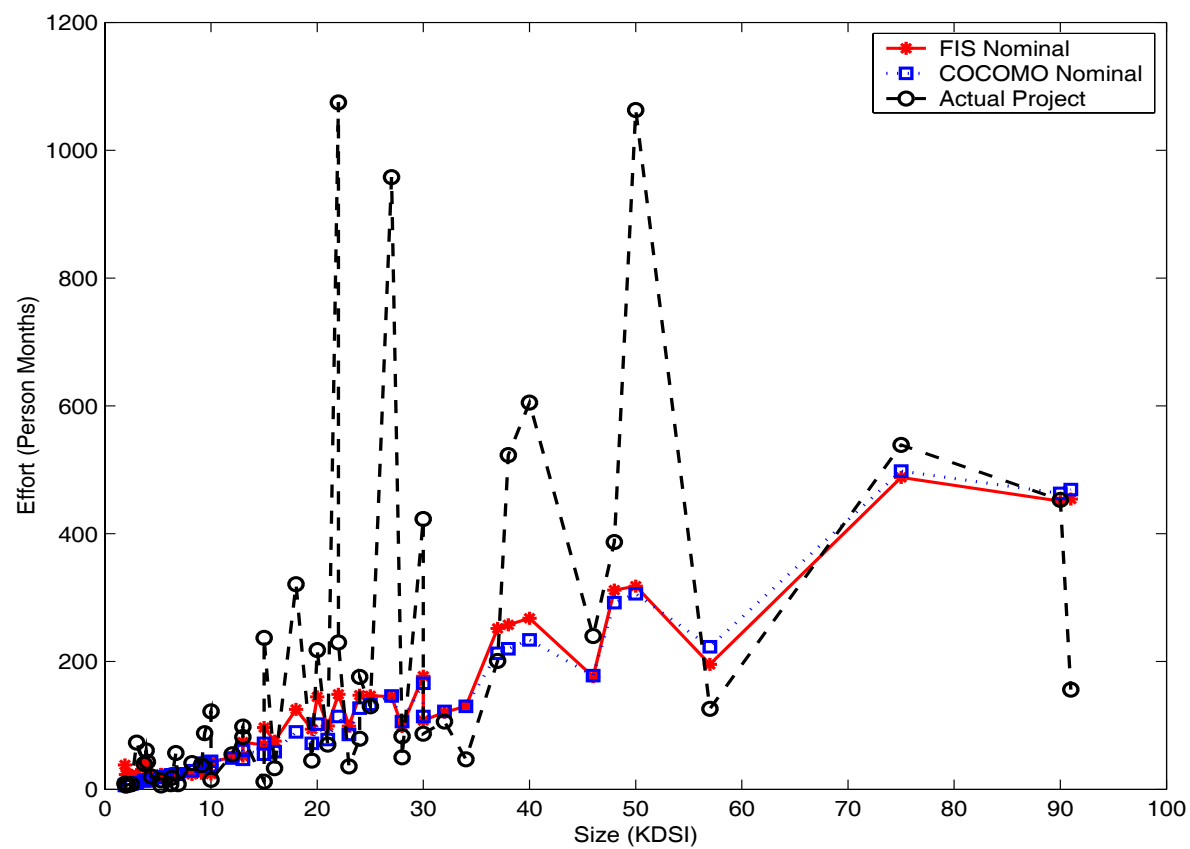
Our validation in this section is carried out in two steps. The first step compares the performance of the trained FIS for nominal effort prediction with that of nominal intermediate COCOMO model estimate on actual project values. The second step compares the performance of trained FIS with the COCOMO model estimates using the same effort multipliers used to adjust the COCOMO model nominal estimates in the database to adjust our predicted effort from trained FIS.

The nominal effort from intermediate COCOMO model has 26% of its predicted values fall within 25% of their actual values ( $PRED(25) = 26\%$ ) on the COCOMO database, while our trained FIS has 23% of its predicted

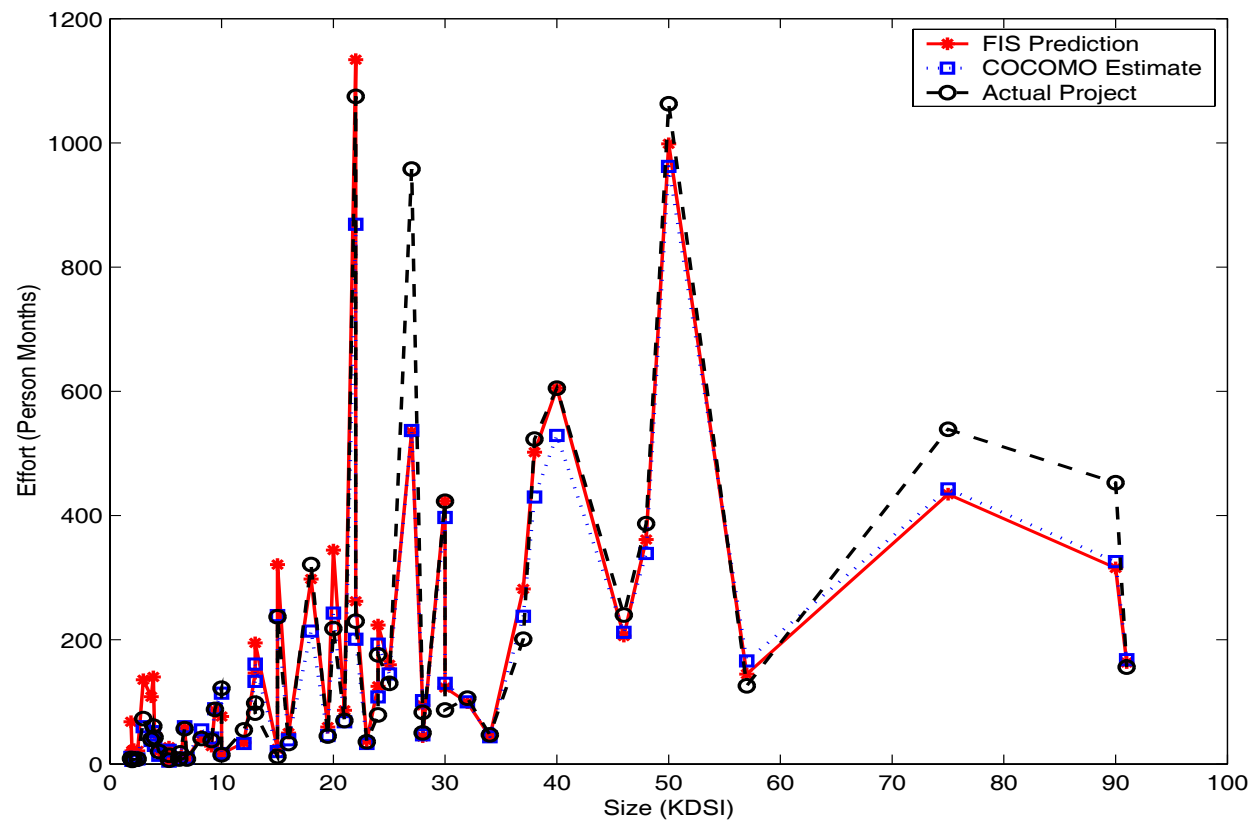
values fall within 25% of their actual values ( $PRED(25) = 23\%$ ). A graph comparing the nominal effort predicted using trained FIS and COCOMO model on actual projects in the database is given in Figure 48.

In the second validation experiments, tuning with effort adjustment factor (EAF) of the cost drivers in the COCOMO database, the intermediate COCOMO model has 72% of the predicted values fall within 25% of their actual values ( $PRED(25) = 72\%$ ). The trained FIS when tuned with EAF has 55% of the predicted values fall within 25% of their actual values ( $PRED(25) = 55\%$ ). The graph in Figure 49 compares the adjusted effort predicted using the trained FIS and COCOMO model on actual projects in the database.



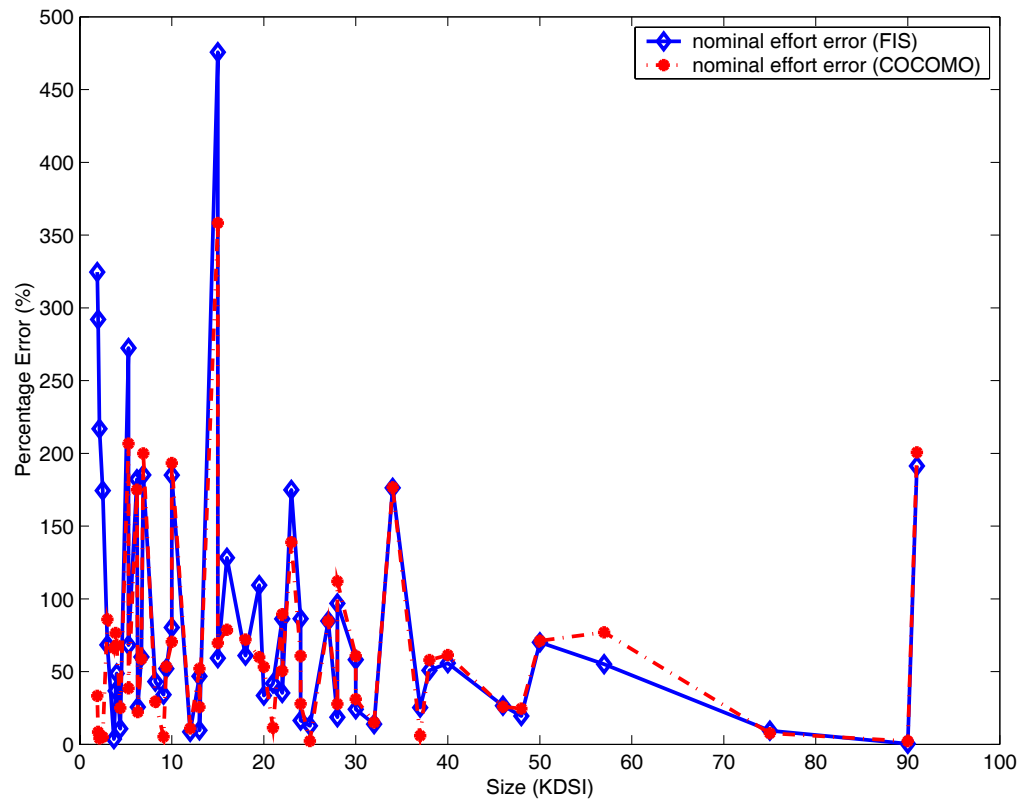


**Figure 48: Nominal Effort Prediction of Trained FIS and COCOMO model on COCOMO database.**

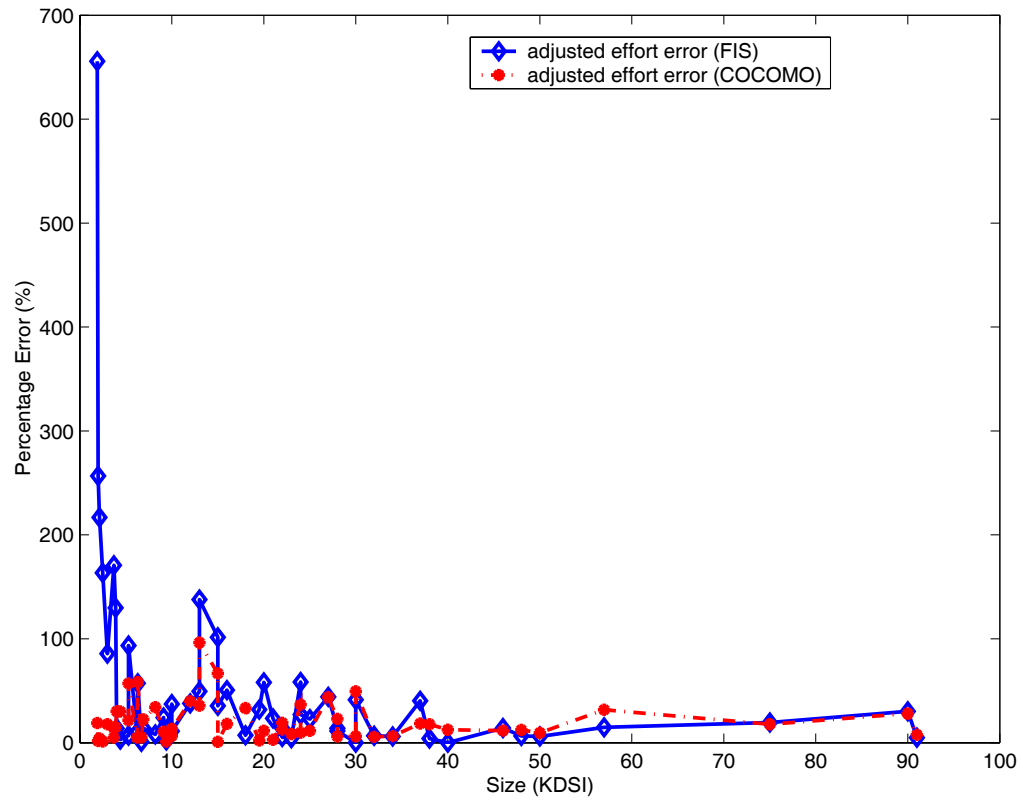


**Figure 49: Effort Prediction of Trained FIS and COCOMO model adjusted by effort multipliers on COCOMO database.**

We plot a graph of the percentage error on predictions made by trained FIS and COCOMO model using the COCOMO database. These graphs are shown in Figure 50 and Figure 51 for the nominal effort and adjusted effort predictions respectively. The discussion of the error graphs follows the figures.



**Figure 50: Percentage error of the nominal effort predictions obtained from trained FIS and COCOMO model using the COCOMO database.**



**Figure 51: Percentage error of the adjusted effort predictions obtained from trained FIS and COCOMO model using the COCOMO database with effort multipliers.**

Considering the percentage error graphs, the trained FIS recorded marginally higher percentage errors on the average, thus making the predictions of the COCOMO model a little better.

In summary, the prediction quality and the percentage errors show that COCOMO model marginally out-performs the trained FIS in this test. This

observation is not surprising since our trained FIS derived most of its initial knowledge from the COCOMO model, including the training data.

#### **6.4.5.2 Validation using the COCOMO Database to Train the FIS**

The aim of the first validation experiment carried out here is to strengthen our assertion that, the availability of more information would enhance the prediction capabilities of the framework.

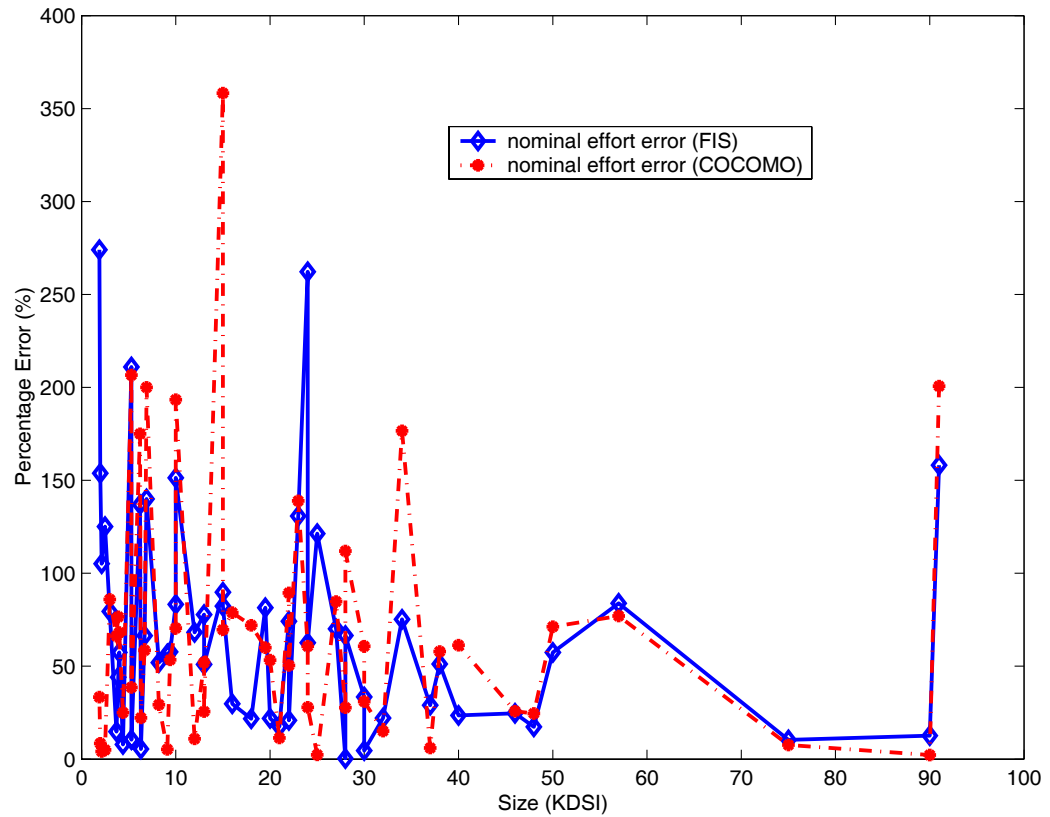
Two validation experiments were carried out here. The first experiment partitioned the 53 data-points selected from the COCOMO database into 39 training and 14 validation datasets. After training, the FIS yields nominal effort with prediction accuracy of  $PRED(25) = 26\%$  on training data,  $PRED(25) = 29\%$  on validation dataset, and  $PRED(25) = 26\%$  on the entire 53 data-points. This shows that both trained FIS and COCOMO model achieved the same prediction quality –  $PRED(25) = 26\%$ .

In the second validation experiment the whole dataset of 53 data-points was used for training, yielding prediction accuracy of  $PRED(25) = 30\%$  after training. The prediction accuracy recorded by the trained FIS is higher than that of COCOMO model which reported  $PRED(25) = 26\%$ , as discussed in the previous section.

Figure 52 shows the percentage prediction error recorded by the trained FIS and COCOMO model when the FIS is trained using all the data points

selected from the COCOMO database. The error graphs reveals that the trained FIS, while it out-performs the COCOMO model with  $PRED(25) = 30\%$  against  $PRED(25) = 26\%$ , the percentage errors of the two are comparable. In sort, the COCOMO model prediction contains the data-point with the largest percentage error, meaning that our trained FIS out-performs the COCOMO model.

In concluding our validation using live project database, the results presented in this section revealed that performance of a trained FIS is comparable to that of the COCOMO model in all ramifications. Since the COCOMO model was developed using the same database, and our trained FIS out-performing it when trained using the database indicates that results that are more promising could be achieved when the FIS is augmented with more information in the form of expert knowledge, which the COCOMO model cannot incorporate.



**Figure 52: Percentage error of the nominal effort predictions obtained from trained FIS and COCOMO model using the COCOMO database for training and testing.**

#### 6.4.6 Experimental V – Validating the Training Algorithms

The basic objective of this last experiment is to buttress our assertion/hypothesis that our training procedure for developing a fuzzy inference system for software effort prediction is worth the investment.

The approach involves developing a badly formulated FIS rulebase and investigates if our training procedure can still rediscover a pattern for improvement. If this objective is realized, then the presence of experts to provide adequate knowledge for building correct rulebase would definitely enable the trained FIS to provide qualitative estimates always.

The procedure to develop a bad rulebase is to distort the rulebase already well formulated to make it less meaningful. Our procedure for achieving this is given in Algorithm 1.7.

**ALGORITHM 1.7** - Building a Bad Rulebase

- 1: Create a fuzzy inference system with a rulebase formulated using the knowledge of relationships between *mode*, *size* and *effort* in the COCOMO model.
- 2: Extract the rules automatically generated in (1:) and save the initial FIS
- 3: Disorganize the rulebase such that they are less meaningful. For example, a combination of organic mode M1, and specific size S2 should give effort C1. Change the conclusion of this particular rule to C2, which is supposed to be the effort for a mode M1 and size S2.
- 4: Repeat steps 1 – 3 until many of the rules are made less meaningful based on the relationships in (1:)
- 5: Update the initial FIS with the updated rulebase and save as FIS2.
- 6: Train the new FIS2 to give a finally trained FIS3
- 7: Evaluate the prediction quality of the trained FIS3, and compare the result of the prediction using the saved FIS2 (5 :)

Table 26 is the rulebase of a developed fuzzy inference system with 3 input MFs for the *mode* input variable, 5 fuzzy sets for the *size* input variable and 15 fuzzy sets for the *effort* output variable. Each row in the initial rulebase



column and the disorganized rulebase column is a fuzzy rule made up of 5 column entries with the following description:

- Column 1- Input 1 (mode)
- Column 2- Input 2 (size)
- Column 3- Output 1 (effort)
- Column 4- Rule weight
- Column 5- Rule conjunction (AND)

The rules whose conclusions are in bold in Table 26 represent the rules distorted.

**Table 26: Rulebase of an FIS and the disorganized version of the same rulebase**

Initial Rulebase	Disorganized Rulebase
1 1 1 1 1	1 1 1 1 1
2 1 2 1 1	2 1 2 1 1
3 1 3 1 1	3 1 3 1 1
1 2 <b>4</b> 1 1	1 2 <b>6</b> 1 1
2 2 5 1 1	2 2 5 1 1
3 2 <b>6</b> 1 1	3 2 <b>4</b> 1 1
1 3 7 1 1	1 3 7 1 1
2 3 8 1 1	2 3 8 1 1
3 3 <b>9</b> 1 1	3 3 <b>12</b> 1 1
1 4 10 1 1	1 4 10 1 1
2 4 <b>11</b> 1 1	2 4 <b>14</b> 1 1
3 4 <b>12</b> 1 1	3 4 <b>15</b> 1 1
1 5 <b>13</b> 1 1	1 5 <b>12</b> 1 1
2 5 <b>14</b> 1 1	2 5 <b>11</b> 1 1
3 5 <b>15</b> 1 1	3 5 <b>13</b> 1 1

For example, row 4 of the initial rulebase contains the rule (1 2 4 1 1) which translates to the following:

IF *MODE* is Organic AND *SIZE* is S2 THEN *EFFORT* is effortMF4      **Equation 17**

In the disorganized rulebase, however, the rule has been changed to the following:

IF *MODE* is Organic AND *SIZE* is S2 THEN *EFFORT* is effortMF6      **Equation 18**

Meanwhile, the rule output MF that forms the consequent of Equation 18 is meant for the following rule:

IF *MODE* is Embedded AND *SIZE* is S2 THEN *EFFORT* is effortMF6      **Equation 19**

After distorting the rulebase, it is expected that the prediction accuracy should drop, and this is actually the case obtained for the FIS with distorted rulebase. The objective now turns to optimizing the distorted rulebase in order to investigate whether our training procedure would be able to discover a better pattern to make some improvements.

The results of training the distorted rulebase, as compared to what we obtained with the trained FIS with rules intact is given in Table 27. In spite

of the fact that our rulebase has been made less meaningful, we still were able to arrive at some improvements after training. In particular, the prediction quality – PRED(25) on testing with the validation data improved from 59.94% to 70.59%, and improves from 54.55% to 62.12% on the training data.

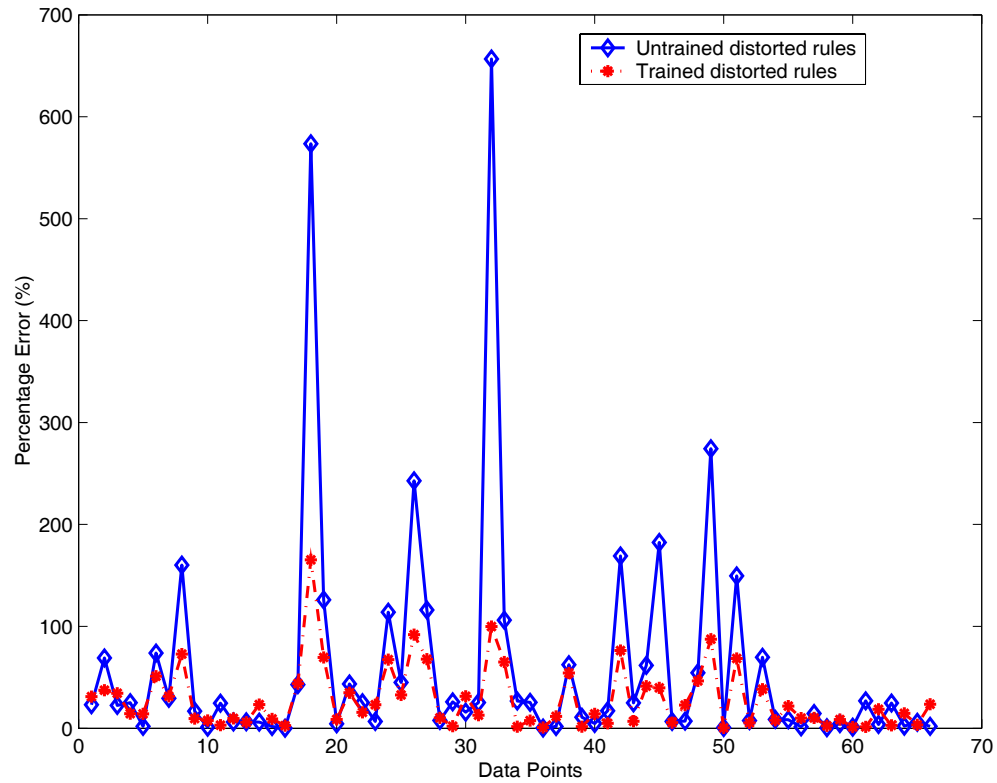
**Table 27: Experimental result of the organized and disorganized rulebases tested on the training data and validation data, before and after training for 56 epochs.**

Type of Data	Original FIS with Rule Base Intact				FIS with Disorganized Rule Base			
	Prediction Accuracy PRED(25) (%)		RMSRE		Prediction Accuracy PRED(25) (%)		RMSRE	
	Before Training	After Training	Before Training	After Training	Before Training	After Training	Before Training	After Training
Validation	76.47	91.18	0.8637	0.1535	52.94	70.59	0.7795	0.2720
Training	77.27	86.36	1.0315	0.2875	54.55	62.12	1.2908	0.4175

It is worth noting, however, that the ability of our training procedure to optimize a badly formulated rulebase with blind knowledge is significant, thus, the availability of an expert to furnish the rulebase with meaningful knowledge would make our training procedure a very reliable and practical approach to effort prediction using soft computing.

In addition, the training has offered considerable reduction in the RMSRE while testing the trained FIS using validation and training datasets. The importance of this little discovery is again significant. It implies that, while about 70.59% predicted values fall within 25% of the actual values, it clear that the bulk of the other data points are very close to the 25% cut, unlike the untrained distorted rulebase FIS that has larger RMSRE.

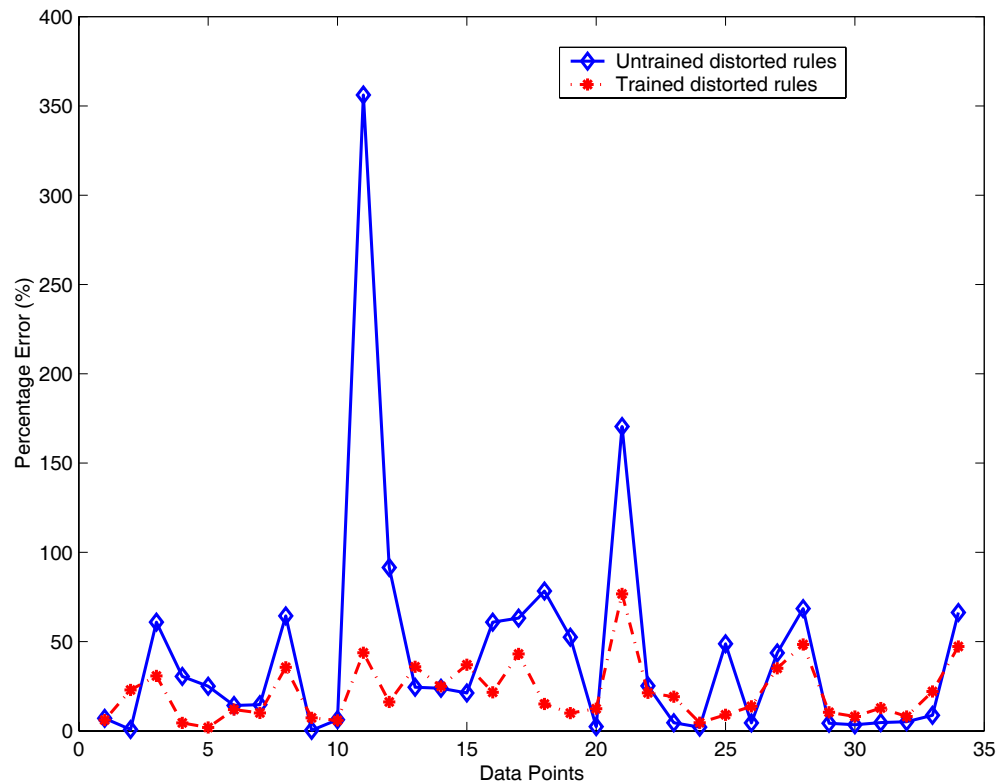
Figure 53 shows the percentage error when testing the untrained FIS and trained FIS with distorted rulebase using the training dataset. From the graph, it can be seen that almost 100% of the prediction made by our trained FIS fall within **100%** of the actual values while the corresponding untrained FIS has 100% of the prediction fall within **700%** of the actual value. The prediction error margin of 100% - 700% is too wide, even as the trained FIS still maintains prediction quality that is very much higher than the untrained FIS.



**Figure 53: Percentage error of the predictions obtained using the trained and untrained FIS with distorted rules on TRAINING DATA**

In the same vein, Figure 54 show similar experimental result on the validation dataset that was not seen during training. Again, our trained FIS would yield almost 100% of predicted values falling within **80%** error margin, while the untrained FIS would give the same 100% prediction at an error margin of about **400%**.

Having carried out the error analysis to support the performance of the training procedure discussed in this thesis research, it is safe to conclude that our training procedure is worth the investment, and is definitely a candidate for use in real project effort prediction to aid software engineering project management.



**Figure 54: Percentage error of the predictions obtained using the trained and untrained FIS with distorted rules on VALIDATION DATA**

This chapter has presented a number of experiments to realize our proposed framework in this thesis. We have explored several alternative routes in order to substantiate any claim made regarding the finally chosen track.

Results from our experiments in this chapter have shown that the move from regression based to soft computing based prediction techniques is justifiable. This justification is based on the fact that, soft computing through the training and adaptation algorithm we have implemented in the framework tolerates imprecision, explains prediction results through rules, incorporates expert knowledge, offers transparency in the prediction system, and could adapt to a new environment as new data becomes available. In adapting after training, only contributing rules to an output of a trained system are affected, the entire knowledge in the fuzzy system is not destroyed because of the modularity in the embedded knowledge.

In addition, the fact is further buttressed by the results of our testing trained FIS against the COCOMO database. The trained FIS offers prediction accuracy close to the COCOMO model when trained using artificial datasets, and offers better prediction accuracy than the COCOMO model when trained using the COCOMO database.

In concluding discussion in this chapter, the significant improvement in performance and reduction in error values obtained from experiments

validating our training procedure, even in the presence of counter-intuitive rulebase, stands to strengthen the approach employed.



## ***CHAPTER 7***

### **CONCLUSION**

#### **7.1 Introduction**

This chapter presents a summary of our major contributions in this thesis work to the software engineering community. It also provides a few suggestions for future research directions.

#### **7.2 Summary of Contributions**

The thesis research has resulted in the following contributions to knowledge:

1. Provided a generic set of attributes (i.e., characteristics) proposed in Chapter 3 to classify soft computing based effort prediction techniques. These characteristics, can serve as a theoretical basis upon which formal assessment metrics can be built. We have presented a rating scheme as a proof-of-concept to justify the potentials of such metrics.
2. Presented an extensive critical survey and theoretical evaluation (guided by the proposed attributes) of the state-of-the-art application of soft computing in development effort prediction.

3. Presented a transparent fuzzy logic based framework, equipped with training and adaptation algorithms for development effort prediction. The framework described, implemented and validated, allows contribution from experts in a manner that would enable the prediction technique model and adapt to the environment of the prediction problem.
4. Demonstrated the capabilities of the framework through empirical validation carried out on artificial datasets and the COCOMO public database of completed projects. The advantages of the framework include the following:
  - a. Tolerance of imprecision
  - b. Incorporating experts knowledge
  - c. Transparency in the prediction system
  - d. Ability to explain prediction results through rules
  - e. Adaptability to new environments as new data becomes available
5. When validated on a dataset of 53 projects, the intermediate COCOMO model yielded a prediction accuracy of  $PRED(25) = 26\%$  for nominal effort and  $PRED(25) = 72\%$  using the effort

adjustment factor. The trained FIS implementing the framework reports prediction accuracy of  $PRED(25) = 23\%$  for nominal effort and  $PRED(25) = 55\%$  with same effort adjustment factor. On training the FIS using real COCOMO database project data, the prediction quality of the trained FIS improved to  $PRED(25) = 30\%$ , which is better than that of the COCOMO model.

6. Reported promising experimental summary results in spite of the fact that, the knowledge in the rulebase and the training data are partly generated artificially with the help of the intermediate COCOMO model. It does signify that there are potentials for improvements when the framework is deployed in practice, since experienced experts could augment it with their knowledge.

### 7.3 Future Research

Some of the openings, which we have pointed to earlier, and that can be investigated in future research include the following:

1. Developing more formal metrics based on our generic set of attributes to evaluate soft computing based prediction systems, would be a potential candidate for further research. In Section 3.3, we suggested an assessment metric that could be a stepping-stone

to developing such formal metrics. We have presented a rating scheme that could be further developed using information from completed software projects and experts judgment to ascertain the impact of each factor on the prediction problem to be estimated.

2. Deploying the framework to COCOMO II environment and comparing performance on live projects data. In Section 4.3.5, we stated that our framework is still valid and applicable to COCOMO II once experts are available to give information required for MFs definition and rulebase development.
3. Investigating other applicable MF types like Gaussian and performance analysis can be pursued as a future research. In Section 4.3.3, we stated that the first MF type that readily comes to mind when modeling features like *size*, *mode* and *effort*, as a fuzzy system is triangular MFs, and we have used this MF type in our experiments.
4. A future research may involve investigating other approaches to normalizing the error measure. In our implementation of the training and adaptation algorithms, we have just used our intuition to normalize the output error from a fuzzy system using the range of the output variable, and we observed improved

performance. The approach we have used shields the user or trainer of the prediction system from worrying about how to normalize dataset before training or using the system.

5. Investigating the performance of other defuzzification techniques may be pursued in a future work. In Section 5.3.2.2, we stated that the heuristic used in modifying the output fuzzy sets considers the defuzzification procedures. In our current implementation, we have used COA and it performs well.

## Bibliography

- [1] Anderson D. and McNeil, G.: “Artificial Neural Networks Technology”, Data & Analysis Center for Software, 775 Daedalian Drive, Rome, NY, August 1992.
- [2] Andreas, N., Nauck, D. and Kruse, R.: “Neuro-Fuzzy Development Tool for Fuzzy Controllers under MATLAB/SIMULINK”, In Proc. Of the 5th European Congress on Intelligent Techniques and Soft Computing, Sept. 8 - 11, 1997, pp. 1029 – 1033.
- [3] Boehm B. W.: “Software Engineering Economics”, Englewood Cliffs, NJ, Prentice-Hall, 1981.
- [4] Boehm B., Abts, C., and Chulani, S.: “Software Development Cost Estimation Approaches – A Survey”, University of Southern California Center for Software Engineering, Technical Reports, USC-CSE-2000-505, 2000.
- [5] Boehm, B. *et al.* : “Cost Models for Future Software Life Cycle Processes: COCOMO 2.0”, Annals of Software Engineering Special Volume on Software Process and Product Measurement, Science Publisher, Amsterdam, The Netherlands, Vol. 1, pp. 45 – 60, 1995.
- [6] Boetticher G. D.: “An Assessment of Metric Contribution in the Construction of a Neural Network-Based Effort Estimator”, Proceedings of 2nd International Workshop on Soft Computing Applied to Software Engineering, 2001.
- [7] Boetticher G. D.: “Using Machine Learning to Predict Project Effort: Empirical Case Studies in Data-Starved Domains”, Proceedings of Model Based Requirements Workshop, San Diego, pp 17 – 24, 2001.
- [8] Briand, L. C. and Wieczorek I.: “Resource Estimation in Software Engineering”, To appear in the second edition of the Encyclopedia of Software Engineering, Wiley, 2001.
- [9] Burgess, C. J. and Lefley M.: “Can Genetic Programming improve software effort estimation? A comparative evaluation”, Information and Software Technology 43 (2001), pp. 863-873.
- [10] Chulani, S., Boehm B., and Steece, B.: “Calibrating Software Cost Models Using Bayesian Analysis”, University of Southern California Center for Software Engineering, Technical Reports, USC-CSE-98-508, 1998.
- [11] Clark, B. K.: “The Effects of Software Process Maturity on Software Development Effort”, PhD Dissertation, Faculty of Graduate School, University of Southern California, Aug. 1997.
- [12] Conte, S., Dunsmore, H. and Shen, V., “Software Engineering Metrics and Models”, Benjamin-Cummings, Menlo Park, CA, 1986.
- [13] Devnani-Chulani, S.: “Bayesian Analysis of Software Cost and Quality Models”, PhD Dissertation, Faculty of Graduate School, University of Southern California, May 1999.

- [14] Dolado J. J.: "On the Problem of the Software Cost Function", *Information and Software Technology*, 43 (2001), pp. 61-72.
- [15] Eklund, P., Klawonn, F., and Nauck, D.: "Distributing Errors in Neural Fuzzy Control", In *Proc. of Intern'l Conf. on Fuzzy Logic and Neural Networks (IIZUKA '92)*, Japan, 1992, pp. 1139-1142.
- [16] Fenton, N.E. and Pfleeger, S.L.: "Software Metrics - A Rigorous and Practical Approach", 2nd Edition, Boston MA, PWS Publishing, 1997.
- [17] Fuzzy Logic Toolbox User's Guide. MATLAB 6.0, Mathworks, Inc., 2000.
- [18] Gray, G. and MacDonell, S.: "Applications of Fuzzy Logic to Software Metric Models Development Effort Estimation". In *Proc. of the 1997 Annual Meeting of the North American Fuzzy Information Processing Society - NAFIPS*, Syracuse NY, USA, IEEE (1997), pp. 394-399.
- [19] Gray, G.R. and MacDonell, S.G.: "Fuzzy Logic for Software Metric Models throughout the Development Life-Cycle". In *Proc. of the 1999 Annual Meeting of the North American Fuzzy Information Processing Society - NAFIPS'99*, IEEE(1999).
- [20] Hodgkinson, A.C. and Garratt, P.W.: "A NeuroFuzzy Cost Estimator". In *Proc. of the 3rd International Conference on Software Engineering and Applications - SAE*, (1999), pp. 401-406.
- [21] Hong, D: "Software Cost Estimation", Department of Computer Science, University of Calgary, Alberta, Canada, <http://pages.cpsc.ucalgary.ca/~hongd/SENG/621/report2.html> (Accessed 2002-07-11).
- [22] Idri, A. and Abran, A.: "A Fuzzy Logic Based Set of Measures for Software Project Similarity: Validation and Possible Improvements", *Proc. of METRICS 2001*, London, England, 2001, pp. 85-96.
- [23] Idri, A. and Abran, A.: "COCOMO Cost Model Using Fuzzy Logic". *7th International Conference on Fuzzy Theory & Technology*, Atlantic City, New Jersey, March 2000.
- [24] Idri, A. and Abran, A.: "Evaluating Software Project Similarity by using Linguistic Quantifier Guided Aggregations", *Proc. of IFSA/NAFIPS 2001*, Vancouver, Canada, 2001, pp. 470 – 475.
- [25] Idri, A. and Abran, A.: "Towards a Fuzzy Logic Based Measures for Software Projects Similarity". *MCSEAI'2000*, Morocco, November 2000.
- [26] Jang, J.-S. R., Sun, C.-T. and Mizutani, E.: "Neuro-Fuzzy and Soft Computing – A Computational Approach to Learning and Machine Intelligence", Prentice-Hall, NJ, 1997.
- [27] Jeffery, D. R. Low, G.C. and Barnes, M.: "A comparison of function point counting techniques" *IEEE Transactions on Software Engineering*, 19 (5), pp 529-532, 1993.
- [28] Kasabov, N. K., Kim, J. S., Gray, A. R. and Watts, M. J. "FuNN – A Fuzzy Neural Network Architecture for Adaptive Learning and

- Knowledge Acquisition” , Information Sciences, Vol. 101, Issue 3, pp. 155 – 175, 1997
- [29] Kirsopp, C. and Shepperd, M.J.: "Making Inferences with Small Numbers of Training Sets", 6th International Conference on Empirical Assessment & Evaluation in Software Engineering, Keele University, Staffordshire, UK, April 8th - 10th, 2002.
  - [30] Kirsopp, C. Shepperd, M.J. and Hart, J.: “Search heuristics, case-based reasoning and software project effort prediction”, Genetic and Evolutionary Computation Conference (GECCO 2002), New York, AAAI, 2002.
  - [31] Kruse, R. and Nurnberger, A.: “Learning Methods for Fuzzy Systems”, Proc. Of 3<sup>rd</sup> German GI-Workshop Fuzzy-Neuro-Systeme’95, Darmstadt, Germany, November 15 – 17, 1995.
  - [32] Kumar, S. Krishna, A. and Satsangi, P.: “Fuzzy systems and neural networks in software engineering project management,” Journal of Applied Intelligence, vol. 4, (1994), pp. 31-52.
  - [33] Lin, C. T.: “Neural Fuzzy Control Systems with Structure and Parameter Learning”, World Scientific Publishing, Singapore, 1994.
  - [34] MacDonell, S.G. and Gray, A.R.: “A Comparison of Modeling Techniques for Software Development Effort Prediction”. In Proc. of the 1997 International Conf. On Neural Information Processing and Intelligent Information Systems, Dunedin, New Zealand, Springer-Verlag (1997) 869-872.
  - [35] MacDonell, S.G. Gray, A. R. and Calvert, M.J.: “FULSOME: A Fuzzy Logic Modeling Tool for Software Metricians”. In Proc. of the 18th International Conference of the North American Fuzzy Information Processing Society – NAFIPS, IEEE (1999), pp. 263-267.
  - [36] MacDonell, S.G. Gray, A. R. and Calvert, M.J.: “FULSOME: A Fuzzy Logic for Software Metric Practitioners and Researchers”. In Proc. of the 6th International Conference on Neural Information Processing – ICONIP, IEEE (1999), pp. 308-313.
  - [37] Mair, C. Kadoda, G. Lefley, M. Phalp, K. Schofield , C. Shepperd, M. Webster, S.: "An Investigation of Machine Learning Based Prediction Systems", Journal of Systems and Software, 53, pp23-29, 2000.
  - [38] Mitra, S. and Hayashi, Y.: “Neuro-Fuzzy Rule Generation: Survey in Soft Computing Framework”, IEEE Transaction on Neural Networks, Vol. 11, No. 3, May 2000.
  - [39] Mitra, S. and Hayashi, Y.: “Neuro-Fuzzy Rule Generation: Survey in Soft Computing Framework”, IEEE Transactions on Neural Networks, Vol. 11, No. 3, May 2000.
  - [40] Mukhopaddhyay, T. and Kekre, S.: “Software Effort Models for Early Estimation of Process Control Applications”. IEEE Transactions on Software Engineering. Vol. 18, No. 10, (1992).



- [41] Musilek, P. Pedrycz, W. Succi, G. and Reformat, M.: "Software Cost Estimation with Fuzzy Models", *Applied Computing Review*, Vol. 8, No. 2, pp 24 – 29, 2000.
- [42] Nauck, D. and Kruse, R.: "A Fuzzy Neural Network Learning Fuzzy Control Rules and Membership Functions by Fuzzy Error Backpropagation", In *Proc. of IEEE Int. Conf. on Neural Networks 1993*, San Francisco, 1993.
- [43] Nauck, D. and Kruse, R.: "NEFCLASS – A Neuro-Fuzzy Approach for the Classification of Data", *Proc. of the ACM Symposium on Applied Computing*, Nashville, Feb. 26-28. ACM Press, 1995.
- [44] Nauck, D. and Kruse, R.: "Training fuzzy sets with neural network method", in *Handbook of Fuzzy Computation*, IOP Publishing Ltd, Philadelphia, pp. D2.6:1-2, 1998.
- [45] Nauck, D., Klawonn, F., and Kruse, R.: "Foundations of Neuro-Fuzzy Systems", Wiley, Chichester, 1997.
- [46] Nauck, D., Rudolf, K. and Stellmach, R.: "New Learning Algorithms for the Neuro-Fuzzy Environment NEFCON-I", In *Proc. of 3<sup>rd</sup> German GI-Workshop "Fuzzy-Neuro-Systeme'95"*, Darmstadt, Germany, Nov. 15-17, 1995, pp. 357-364.
- [47] Nauck, D.: "A Fuzzy Perceptron as a Generic Model for Neuro-Fuzzy Approaches", In *Proc. of Fuzzy-Systeme'94, 2<sup>nd</sup> GI-Workshop*, Munich, Semen Corporation, October 1994.
- [48] Nauck, D.: "Data Analysis with Neuro-Fuzzy Methods", *Habilitation Thesis*, University of Magdeburg, 2000.
- [49] Negnevitsky, M.: "Artificial Intelligence – A Guide to Intelligent Systems", First Edition, Addison-Wesley, England, 2002
- [50] Oller, Albert: "Fuzzy Logic"  
[http://www.etse.urv.es/~aoller/fuzzy/fuzzy\\_logic.htm](http://www.etse.urv.es/~aoller/fuzzy/fuzzy_logic.htm) (Accessed 2002-11-27).
- [51] Pedrycz, W., Peters, H. F., and Ramanna, S.: "A Fuzzy Set Approach to Cost Estimation of Software Projects", *Proc. of IEEE Canadian Conference on Electrical and Computer Engineering*, Alberta, Canada, May 9-12, 1999.
- [52] Pfleeger, S. L.: "Model of Software Effort and Productivity", *Information and Software Technology*, vol. 33, no. 3, April 1991, pp. 224-231
- [53] Putnam, L. H.: "A General Empirical Solution to the Macro Software Sizing and Estimating Problem", *IEEE Transactions on Software Engineering*, Vol. 4, No. 4, pp. 345 – 361, 1978.
- [54] Ramalho, Maria F. N.: "Application of an Automatically Designed Fuzzy Logic Decision Support System to Connection Admission Control in ATM Networks", *PhD Dissertation submitted to the Department of Electronic Engineering, Queen Mary and Westfield College, University of London*, 10 December 1996.

- [55] Razaz, M. and King, J.: "Introduction to Fuzzy Logic". Information Systems – Signal and Image Processing Group. <http://www.sys.uea.ac.uk/~king/restricted/boards/> (Accessed 2003-01-04).
- [56] Rich, E. and Knight, K.: "Artificial Intelligence", 2<sup>nd</sup> Edition, McGraw-Hill, Singapore, 1991.
- [57] Russell, S. J. and Norvig, P.: "Artificial Intelligence: A Modern Approach", 2<sup>nd</sup> Edition, Prentice Hall, 2003.
- [58] Ryder, J.: "Fuzzy Modeling of Software Effort Prediction", Proc. of IEEE Information Technology Conference, Syracuse, NY, 1998.
- [59] Saliu, M. O. and Ahmed M.: "Soft Computing Based Effort Prediction Systems – A Survey", Soft Computing in Software Engineering – Theory and Applications, Edited by E. Damiani and L.C. Jain, Springer-Verlag Publisher, 2003.
- [60] Samson, B., Ellison, D., and Dugard, P. : "Software Cost Estimation Using Albus Perceptron (CMAC)", Information and Software Technology, 39 (1997), pp. 55-60.
- [61] Schofield C.: "Non-Algorithmic Effort Estimation Techniques" , Technical Reports, Department of Computing, Bournemouth University, England, TR98-01, March 1998
- [62] Shepperd M., Kadoda G.: "Using Simulation to Evaluate Prediction Techniques", IEEE Trans. on Softw. Eng. 27(11), pp987-998, 2001.
- [63] Shi, Y., Mizumoto, M., Yubazaki, N. And Otani, M.: "A Learning Algorithm for Tuning Fuzzy Rules Based on the Gradient Descent Method", Proc. of Fifth IEEE Int. Conf. on Fuzzy System, New Orleans, Sept. 8-11, 1996, pp. 55-61.
- [64] Shukla, K. K.: "Neuro-Genetic prediction of software development effort", Information and Software Technology Journal, 42 (2000) pg. 701 – 713.
- [65] Snell, D.: "Software Cost Estimation Website (SCEW)", <http://www.ecfc.u-net.com/cost/compare.htm>, Bournemouth University, 1997
- [66] Sowel, Thomas: "Fuzzy Logic for Just Plain Folks", An online free book, <http://www.fuzzy-logic.com/> (Accessed 2002-11-27).
- [67] Srinivasan, K. and Fisher D.: "Machine Learning Approaches to Estimating Software Development Effort", IEEE Transactions on Software Engineering, Vol.21, No. 2, 1995.
- [68] Strike, K., El-Emam, K. and Madhavji, N.: "Software Cost Estimation with Incomplete Data", IEEE Transactions on Software Engineering, Vol. 27, No. 10, Oct. 2001.
- [69] Stutzke R.D.: "Software Estimating Technology: A Survey", IEEE Software Engineering Project Management, 1997.
- [70] Sullivan, M.: "Genetic Algorithms: A Brief Introduction", [http://www.cs.qub.ac.uk/~M.Sullivan/ga/ga\\_index.html](http://www.cs.qub.ac.uk/~M.Sullivan/ga/ga_index.html), January, 1997.

- [71] Venkatachalam, A. R.: "Software cost estimation using artificial neural networks," In Proceedings of the 1993 International Joint Conference on Neural Networks, (1993), pp. 987-990.
- [72] Wang, L.-X. and Mendel, J. M.: "Generating Fuzzy Rules by Learning from Examples", IEEE Transactions on System, Man, and Cybernetics, Vol. 22, No. 6, 1992.
- [73] Wang, Li-Xin: "Adaptive fuzzy systems and control: design and stability analysis", Prentice Hall, 1994.
- [74] Wittig, G. and Finnie G.: "Estimating software development effort with connectionist models", Information and Software Technology, 39 (1997), pp. 469 – 476.
- [75] Zadeh, L. A.: "The Future of Soft Computing", In Joint 9th IFSA World Congress and 20th NAFIPS International Conference, Vancouver, Canada, July 25 – 28, 2001.
- [76] Zimmermann, H.: "Fuzzy Set Theory – And Its Applications", Kluwer Academic Publishers, Third Edition, 1996.
- [77] Zonglian, F. and Xihui L.: "f-COCOMO: Fuzzy Constructive Cost Model in Software Engineering", Proc. of IEEE Int. Conf. On Fuzzy Systems, IEEE (1992), pp. 331-337.

## Index

### A

Adaptive fuzzy systems, 28

Algorithmic Models

COCOMO, 2, 3, 4, 6, 7, 8, 9, 41, 49, 51,  
53, 54, 55, 59, 60, 62, 65, 68, 71, 72,  
73, 74, 75, 76, 77, 79, 80, 81, 82, 83,  
85, 89, 90, 94, 99, 100, 110, 111, 112,  
130, 132, 182, 183, 184, 185, 186, 187,  
188, 189, 190, 191, 192, 199, 202, 203,  
204, 206, 207, 211, 219

function points, 59

SLIM, 2, 3, 41

### B

Backpropagation, 31, 32, 57, 59, 66, 104

### C

Classification Attributes, 39

attributes importance, 48

prediction system attributes, 41

rating scheme, 46

COCOMO

basic cocomo, 51, 53

COCOMO II, 7, 60, 75, 99, 100, 204

cost drivers, 6, 8, 9, 54, 55, 69, 72, 75, 76,  
78, 79, 87, 88, 90, 91, 92, 94, 95, 97,  
98, 100, 107, 125, 129, 130, 184

detailed cocomo, 9

development mode, 8, 72, 116

intermediate cocomo, 72

organic, 8

scaling factor, 8, 72, 96, 99

semi-detached, 73

Cost drivers, 6, 8, 9, 54, 55, 69, 72, 75, 76, 78,  
79, 87, 88, 90, 91, 92, 94, 95, 97, 98, 100,  
107, 125, 129, 130, 184

CD categorization for fuzzification, 88

Cost estimation

effort prediction, 1, 206, 207, 209, 210

### D

Defuzzification Methods

COA, 23, 24, 25, 26, 27, 119, 205

MOM, 24, 25, 26, 27, 119

### E

Effort Multipliers. *see* cost drivers

Estimation techniques

algorithmic, 3

non-algorithmic, 4

Evolutionary computation

evolutionary strategies, 32

genetic algorithms, 33

genetic programming, 35, 206

Experiments

experimental design, 125

### F

FIS. *see* fuzzy inference system

Framework

framework, 70, 77, 99, 107, 110, 208

fuzzy adjustment, 87

fuzzy COCOMO, 79

parameter learning, 110, 115, 208

rules learning, 104

sensitivity analysis, 94

structure learning, 110

training algorithms, 110, 132, 191

training approach, 101, 107

Fuzzy inference matrix. *see* fuzzy inference

system

Fuzzy system architecture, 20

defuzzification, 23

fuzzification, 22, 69, 98

## G

Generic fuzzy perceptron, 101, 103, 104, 105,  
108, 109, 209

NEFCLASS, 105, 209

NEFCON, 105, 106, 107, 114, 209

NEFPROX, 105, 107

## I

Input Membership Functions

shouldered MFs, 133, 151

un-shouldered MFs, 143, 158

## K

KDSI, 6, 8, 59, 72, 82, 83, 95, 183, 219

## M

Mamdani fuzzy reasoning, 22, 108, 109, 116,  
118, 122

Membership functions, 24, 25, 56, 80, 83, 91,  
92, 94, 101, 102, 103, 109, 110, 112, 113,  
114, 115, 119, 127, 128, 129, 130, 133,  
143, 151, 158, 165, 180

## N

Neural networks

neuron, 29

perceptron, 30, 210

Neuro-Fuzzy, 5, 62, 101, 102, 206, 207, 208,  
209

Neuro-Genetic, 65, 210

Non-Algorithmic Techniques

machine learning, xv, 2, 32

regression trees, 2

## P

prediction accuracy, 132

prediction problem, 9, 39, 48, 49, 202, 204

Prediction quality. *see* prediction accuracy

prediction technique, xv, 9, 39, 69, 199, 201,  
202

## R

reinforcement learning, 107

## S

Soft computing

evolutionary computation, 12, 32, 61, 208

fuzzy logic

fuzzy inference, xv, 4, 5, 6, 7, 10, 12,  
15, 18, 19, 20, 27, 28, 40, 43, 44, 53,  
55, 69, 71, 79, 102, 103, 107, 129,  
202

neural networks, 12, 28, 49, 57, 59, 206,  
207, 208, 209, 211

supervised learning, 107

**V**

## Validation

COCOMO database, 82, 83, 182, 183, 189

## Appendix A: DATASETS

### ARTIFICIAL DATASET AUTOMATICALLY GENERATED (100 DATAPOINTS)

Mode	Size	Effort	1.2000	31.3900	175.1100	1.2000	66.3200	429.6700
1.2000	27.3800	148.6200	1.1200	51.2500	246.5900	1.1200	55.4500	269.3300
1.2000	86.5600	591.4800	1.2000	99.0500	695.3200	1.2000	75.5600	502.4700
1.1200	16.0400	67.1400	1.1200	2.6000	8.7500	1.0500	48.2500	187.4200
1.2000	56.5300	354.7300	1.2000	84.9000	577.8900	1.0500	60.1700	236.3200
1.2000	33.5100	189.3900	1.1200	11.4200	45.8900	1.2000	81.5200	550.4000
1.2000	12.5500	58.2800	1.2000	78.3100	524.4900	1.2000	50.6100	310.6300
1.1200	75.9000	382.8200	1.1200	40.5400	189.6500	1.1200	30.1200	135.9700
1.0500	90.5700	363.0600	1.2000	56.5800	355.1000	1.1200	56.2200	273.5300
1.0500	59.2700	232.6100	1.0500	94.1500	378.1500	1.0500	78.4400	312.1900
1.1200	15.5000	64.6100	1.2000	9.3200	40.7800	1.1200	82.1200	418.1200
1.2000	27.6900	150.6400	1.1200	68.3500	340.4300	1.0500	14.1300	51.6200
1.0500	14.7000	53.8100	1.1200	12.2000	49.4100	1.0500	97.6100	392.7500
1.2000	77.7400	519.9100	1.2000	46.9500	283.8700	1.0500	40.8400	157.3200
1.1200	46.3400	220.2900	1.2000	28.7200	157.3900	1.2000	44.6400	267.1900
1.2000	69.0100	450.6600	1.2000	10.1200	45.0200	1.1200	22.1000	96.1300
1.2000	12.1900	56.2800	1.2000	42.8000	254.0300	1.2000	5.9900	23.9900
1.1200	98.1100	510.3200	1.0500	30.4000	115.3900	1.1200	54.1700	262.3800
1.0500	14.0500	51.3100	1.2000	47.6800	289.1800	1.1200	39.5100	184.2600
1.0500	45.4400	175.9800	1.0500	10.6300	38.2800	1.0500	37.9200	145.5300
1.1200	85.1600	435.5000	1.1200	12.0600	48.7800	1.0500	57.6600	225.9800
1.1200	74.5200	375.0300	1.2000	20.5700	105.4500	1.1200	68.7100	342.4400
1.2000	20.2600	103.5400	1.2000	67.1900	436.4400	1.1200	89.5200	460.5400
1.0500	38.1000	146.2600	1.1200	77.4200	391.4100	1.2000	52.8400	327.1200
1.1200	67.3400	334.8000	1.0500	87.3600	349.5700	1.0500	9.5500	34.2100
1.1200	44.6900	211.5200	1.0500	3.2400	11.0000	1.1200	49.3600	236.4300
1.0500	49.5700	192.8100	1.1200	12.7400	51.8700	1.0500	15.4400	56.6500

1.1200	36.4600	168.4100
1.2000	35.7200	204.4800
1.1200	86.4000	442.6100
1.1200	75.0800	378.1900
1.2000	86.3900	590.0800
1.0500	61.0000	239.7400
1.2000	78.3800	525.0500
1.0500	67.3100	265.8500
1.0500	97.9500	394.1900
1.1200	37.1600	172.0300
1.1200	40.2100	187.9200
1.1200	4.5300	16.2900
1.0500	70.4900	279.0500
1.2000	66.0300	427.4100
1.2000	84.5700	575.2000
1.2000	68.7800	448.8600
1.0500	59.0200	231.5800
1.1200	26.0600	115.6100
1.1200	81.7300	415.9000
1.1200	16.2700	68.2100



**TRAINING DATASET RADOMLY EXTRACTED ARTIFICIAL DATASET  
(66 DATAPOINTS)**

<b>Mode</b>	<b>Size</b>	<b>Effort</b>	1.1200	68.3500	340.4300	1.0500	37.9200	145.5300
1.2000	27.3800	148.6200	1.1200	12.2000	49.4100	1.1200	68.7100	342.4400
1.2000	33.5100	189.3900	1.2000	28.7200	157.3900	1.2000	52.8400	327.1200
1.1200	75.9000	382.8200	1.0500	10.6300	38.2800	1.0500	9.5500	34.2100
1.0500	90.5700	363.0600	1.1200	12.0600	48.7800	1.1200	49.3600	236.4300
1.0500	59.2700	232.6100	1.2000	20.5700	105.4500	1.0500	15.4400	56.6500
1.1200	15.5000	64.6100	1.2000	67.1900	436.4400	1.1200	36.4600	168.4100
1.2000	27.6900	150.6400	1.1200	77.4200	391.4100	1.2000	35.7200	204.4800
1.0500	14.7000	53.8100	1.0500	87.3600	349.5700	1.1200	86.4000	442.6100
1.2000	77.7400	519.9100	1.0500	3.2400	11.0000	1.2000	86.3900	590.0800
1.1200	46.3400	220.2900	1.1200	12.7400	51.8700	1.0500	61.0000	239.7400
1.2000	69.0100	450.6600	1.2000	66.3200	429.6700	1.2000	78.3800	525.0500
1.2000	20.2600	103.5400	1.2000	75.5600	502.4700	1.0500	67.3100	265.8500
1.0500	38.1000	146.2600	1.0500	48.2500	187.4200	1.1200	40.2100	187.9200
1.1200	67.3400	334.8000	1.0500	60.1700	236.3200	1.0500	70.4900	279.0500
1.1200	44.6900	211.5200	1.2000	50.6100	310.6300	1.2000	66.0300	427.4100
1.0500	49.5700	192.8100	1.1200	30.1200	135.9700	1.2000	84.5700	575.2000
1.2000	99.0500	695.3200	1.1200	56.2200	273.5300	1.2000	68.7800	448.8600
1.1200	2.6000	8.7500	1.0500	78.4400	312.1900	1.0500	59.0200	231.5800
1.1200	11.4200	45.8900	1.0500	14.1300	51.6200	1.1200	26.0600	115.6100
1.1200	40.5400	189.6500	1.0500	97.6100	392.7500	1.1200	81.7300	415.9000
1.2000	56.5800	355.1000	1.2000	44.6400	267.1900			
1.0500	94.1500	378.1500	1.2000	5.9900	23.9900			

**VALIDATION DATASET RADOMLY EXTRACTED ARTIFICIAL  
DATASET  
(34 DATAPOINTS)**

<b>Mode</b>	<b>Size</b>	<b>Effort</b>	1.2000	9.3200	40.7800	1.1200	51.2500	246.5900
1.1200	37.1600	172.0300	1.1200	75.0800	378.1900	1.2000	12.5500	58.2800
1.1200	82.1200	418.1200	1.1200	22.1000	96.1300	1.2000	81.5200	550.4000
1.2000	46.9500	283.8700	1.1200	74.5200	375.0300	1.2000	56.5300	354.7300
1.0500	30.4000	115.3900	1.2000	47.6800	289.1800	1.1200	16.0400	67.1400
1.0500	97.9500	394.1900	1.2000	42.8000	254.0300	1.1200	54.1700	262.3800
1.1200	89.5200	460.5400	1.2000	10.1200	45.0200	1.0500	40.8400	157.3200
1.2000	78.3100	524.4900	1.2000	12.1900	56.2800	1.1200	55.4500	269.3300
1.2000	31.3900	175.1100	1.0500	57.6600	225.9800	1.1200	39.5100	184.2600
1.0500	45.4400	175.9800	1.0500	14.0500	51.3100	1.2000	86.5600	591.4800
1.1200	85.1600	435.5000	1.1200	98.1100	510.3200	1.1200	16.2700	68.2100
1.1200	4.5300	16.2900	1.2000	84.9000	577.8900			

**EXTRACTED DATASET (SIZES  $\leq 100$  KDSI) FROM THE COCOMO  
DATABASE OF COMPLETED SOFTWARE PROJECTS  
(53 DATAPOINTS)**

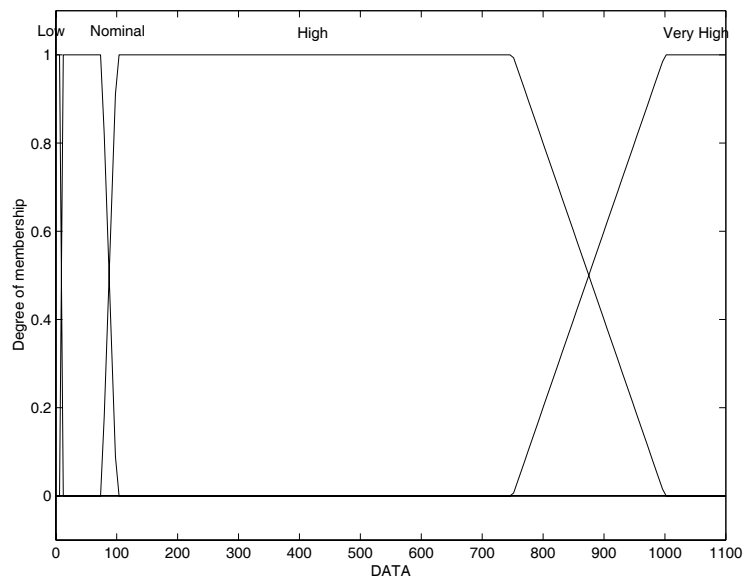
<b>Mode</b>	<b>Size (KDSI)</b>	<b>COCOMO Nominal</b>	<b>Effort Multipliers</b>	<b>COCOMO Adjusted</b>	<b>Actual Project</b>
0001.05	0046.00	0178.00	0001.17	0212.00	0240.00
0001.05	0016.00	0059.00	0000.66	0039.00	0033.00
0001.05	0004.00	0013.70	0002.22	0030.00	0043.00
0001.05	0006.90	0024.00	0000.40	0009.80	0008.00
0001.20	0022.00	0114.00	0007.62	0869.00	1075.00
0001.20	0030.00	0166.00	0002.39	0397.00	0423.00
0001.20	0018.00	0090.00	0002.38	0214.00	0321.00
0001.20	0020.00	0102.00	0002.38	0243.00	0218.00
0001.20	0037.00	0213.00	0001.12	0238.00	0201.00
0001.20	0024.00	0127.00	0000.85	0108.00	0079.00
0001.12	0003.00	0010.30	0005.86	0060.00	0073.00
0001.20	0003.90	0014.30	0003.63	0052.00	0061.00
0001.20	0003.70	0013.50	0002.81	0038.00	0040.00
0001.20	0001.90	0006.00	0001.78	0010.70	0009.00
0001.20	0075.00	0498.00	0000.89	0443.00	0539.00
0001.12	0090.00	0463.00	0000.70	0326.00	0453.00
0001.20	0038.00	0220.00	0001.95	0430.00	0523.00
0001.20	0048.00	0292.00	0001.16	0339.00	0387.00
0001.20	0009.40	0041.00	0002.04	0089.00	0088.00
0001.05	0013.00	0047.00	0002.81	0133.00	0098.00
0001.12	0002.14	0007.00	0001.00	0007.00	0007.30
0001.12	0001.98	0006.40	0000.91	0005.80	0005.90
0001.20	0050.00	0306.00	0003.14	0962.00	1063.00
0001.20	0040.00	0234.00	0002.26	0529.00	0605.00
0001.20	0022.00	0114.00	0001.76	0201.00	0230.00
0001.20	0013.00	0061.00	0002.63	0161.00	0082.00
0001.12	0012.00	0049.00	0000.68	0033.00	0055.00
0001.05	0034.00	0130.00	0000.34	0044.00	0047.00
0001.05	0015.00	0055.00	0000.35	0020.00	0012.00
0001.05	0006.20	0022.00	0000.39	0008.40	0008.00
0001.05	0002.50	0008.40	0000.96	0008.10	0008.00
0001.05	0005.30	0018.40	0000.25	0004.70	0006.00
0001.05	0019.50	0072.00	0000.63	0046.00	0045.00
0001.05	0028.00	0106.00	0000.96	0102.00	0083.00
0001.05	0030.00	0114.00	0001.14	0130.00	0087.00
0001.05	0032.00	0122.00	0000.82	0100.00	0106.00
0001.05	0057.00	0223.00	0000.74	0166.00	0126.00

0001.05	0023.00	0086.00	0000.38	0033.00	0036.00
0001.12	0091.00	0469.00	0000.36	0168.00	0156.00
0001.20	0024.00	0127.00	0001.52	0193.00	0176.00
0001.05	0010.00	0036.00	0003.18	0114.00	0122.00
0001.05	0008.20	0029.00	0001.90	0055.00	0041.00
0001.12	0005.30	0019.40	0001.15	0022.00	0014.00
0001.05	0004.40	0015.00	0000.93	0014.00	0020.00
0001.05	0006.30	0022.00	0000.34	0007.50	0018.00
0001.20	0027.00	0146.00	0003.68	0537.00	0958.00
0001.20	0015.00	0072.00	0003.32	0239.00	0237.00
0001.20	0025.00	0133.00	0001.09	0145.00	0130.00
0001.05	0021.00	0078.00	0000.87	0068.00	0070.00
0001.05	0006.70	0023.60	0002.53	0060.00	0057.00
0001.05	0028.00	0106.00	0000.45	0047.00	0050.00
0001.12	0009.10	0036.00	0001.15	0042.00	0038.00
0001.20	0010.00	0044.00	0000.39	0017.00	0015.00

## Appendix B: COST DRIVERS MFs

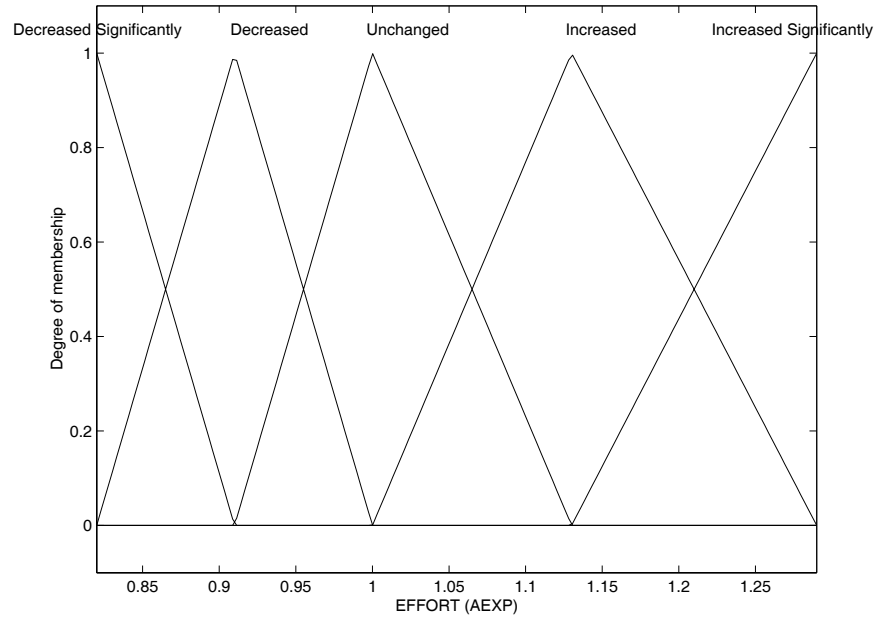
### 1. DATA – Database Size (P-Size SLOC, D- Database size)

Low	Nominal	High	Very High
$D/P < 10$	$10 \leq D/P < 100$	$100 \leq D/P < 1000$	$D/P \geq 1000$



**Figure 55: Antecedent MFs for the FIS of DATA Cost driver**

Low	Nominal	High	Very High
0.94	1.00	1.08	1.16



**Figure 56: Consequent MFs for the FIS of DATA Cost driver**

RULES:

1. If (DATA is Low) then (EFFORT is Decreased)
2. If (DATA is Nominal) then (EFFORT is Unchanged)
3. If (DATA is High) then (EFFORT is Increased)
4. If (DATA is Very High) then (EFFORT is Increased Dramatically)

2. TURN – Computer Turnaround Time

Low	Nominal	High	Very High
Interactive	Turnaround < 4 hours	4-12 hours	> 12 hours

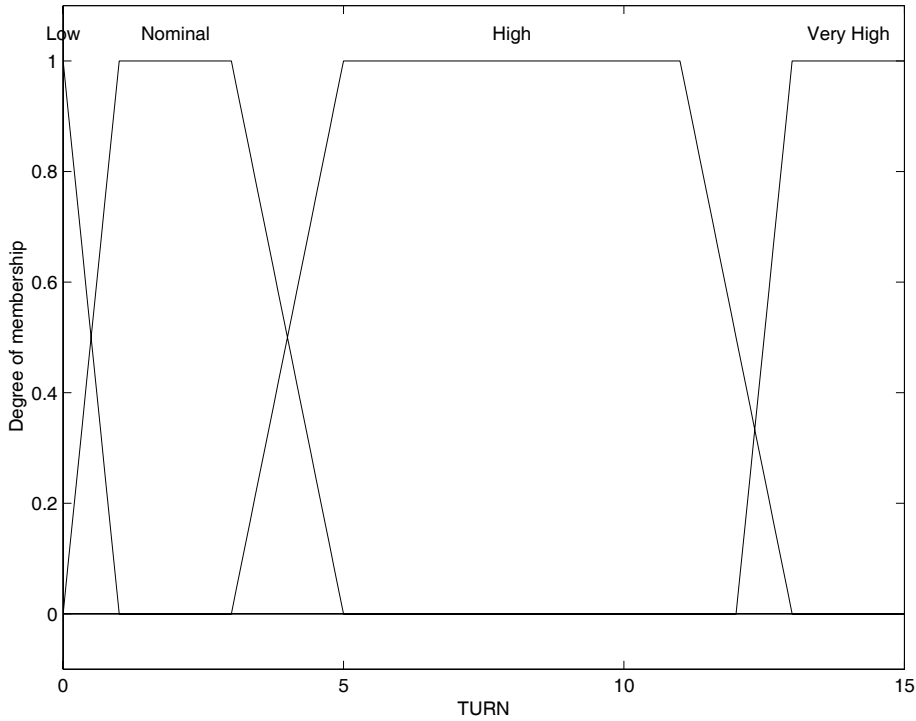
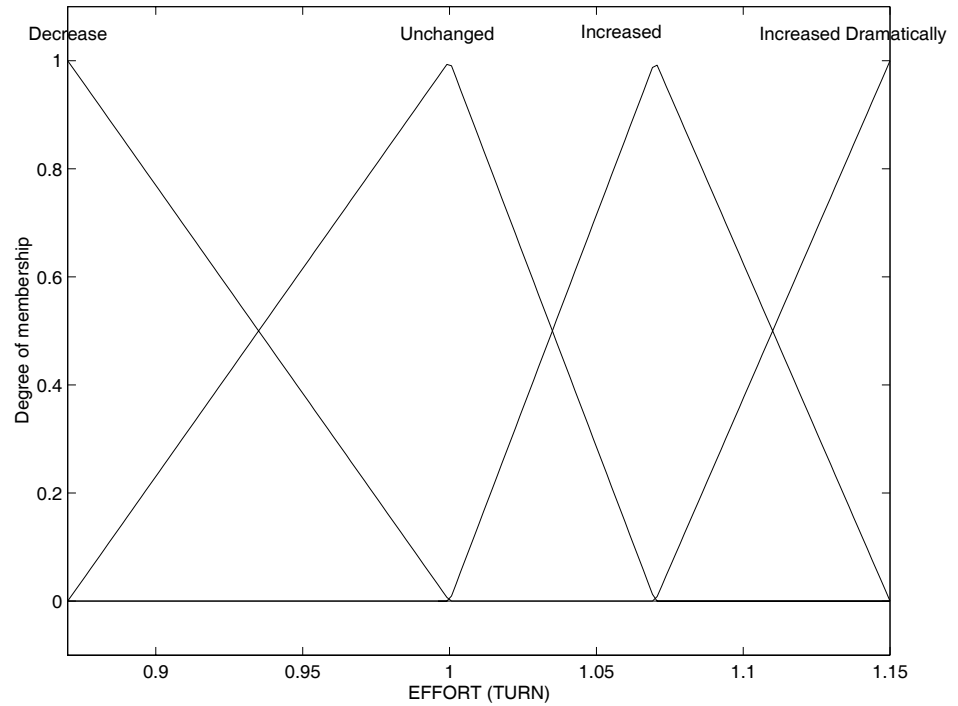


Figure 57: Antecedents MFs for the FIS of TURN Cost driver

Low	Nominal	High	Very High
0.87	1.00	1.07	1.15



**Figure 58: Consequent MFs for the FIS of TURN Cost driver**

RULES:

1. If (TURN is Low) then (EFFORT is Decreased)
2. If (TURN is Nominal) then (EFFORT is Unchanged)
3. If (TURN is High) then (EFFORT is Increased)
4. If (TURN is Very High) then (EFFORT is Increased Dramatically)



3. TIME – Computer Turnaround Time (Use of available execution time)

Nominal	High	Very High	Extra High
≤ 50% use	70%	85%	95%

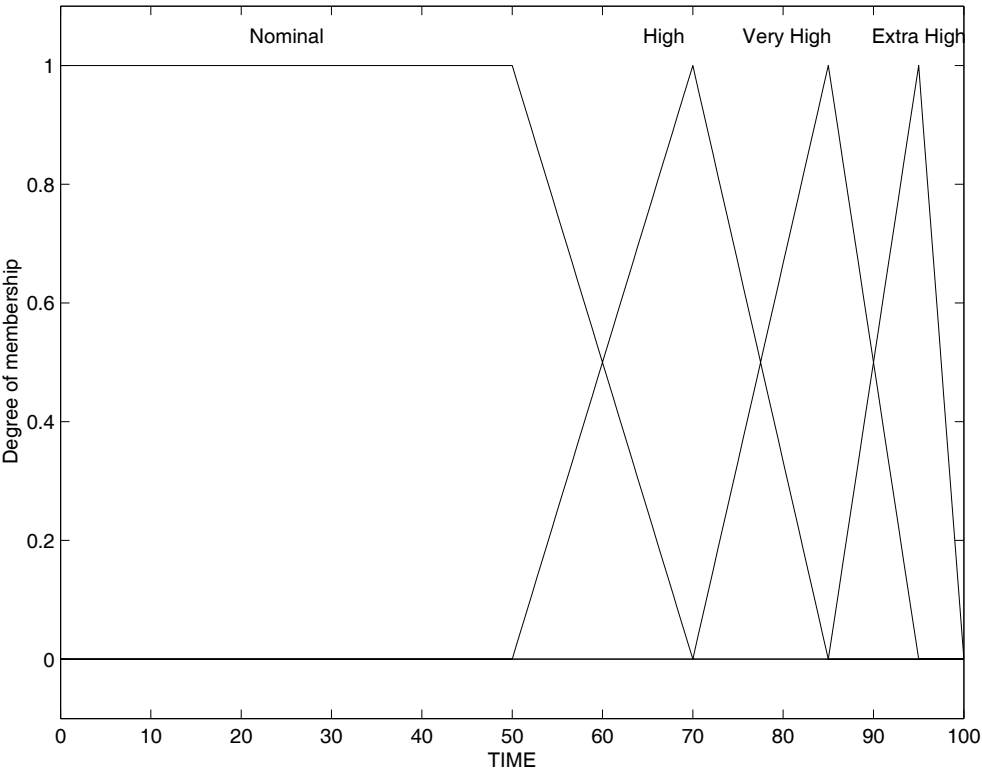
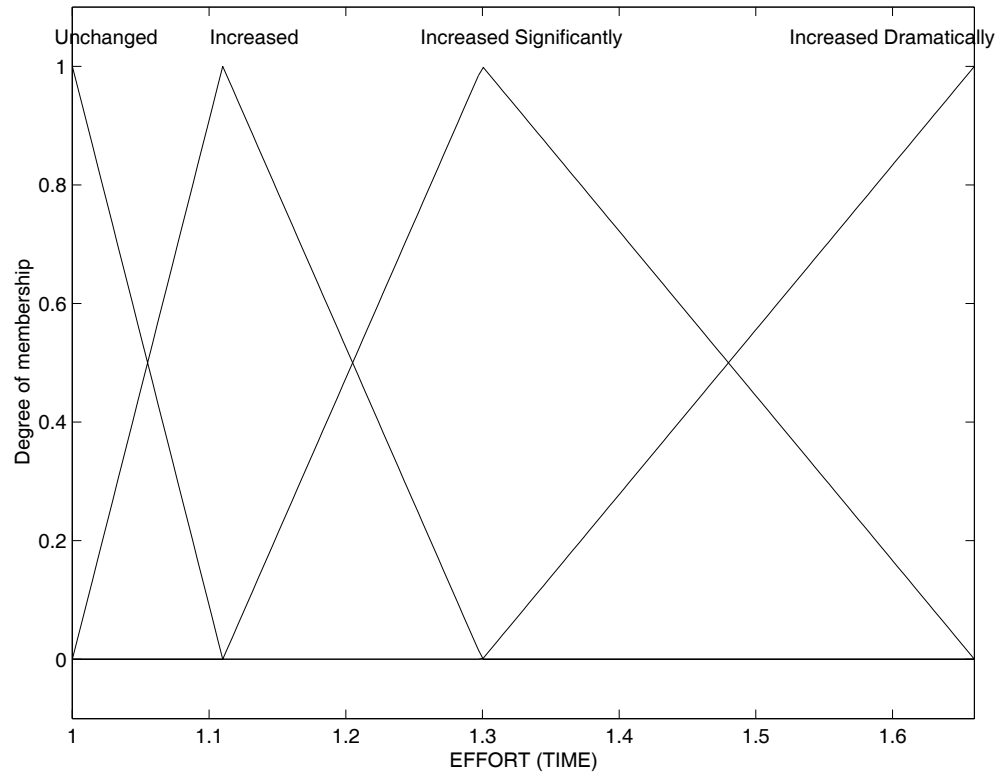


Figure 59: Antecedent MFs for the FIS of TIME Cost driver

Nominal	High	Very High	Extra High
1.00	1.11	1.30	1.66



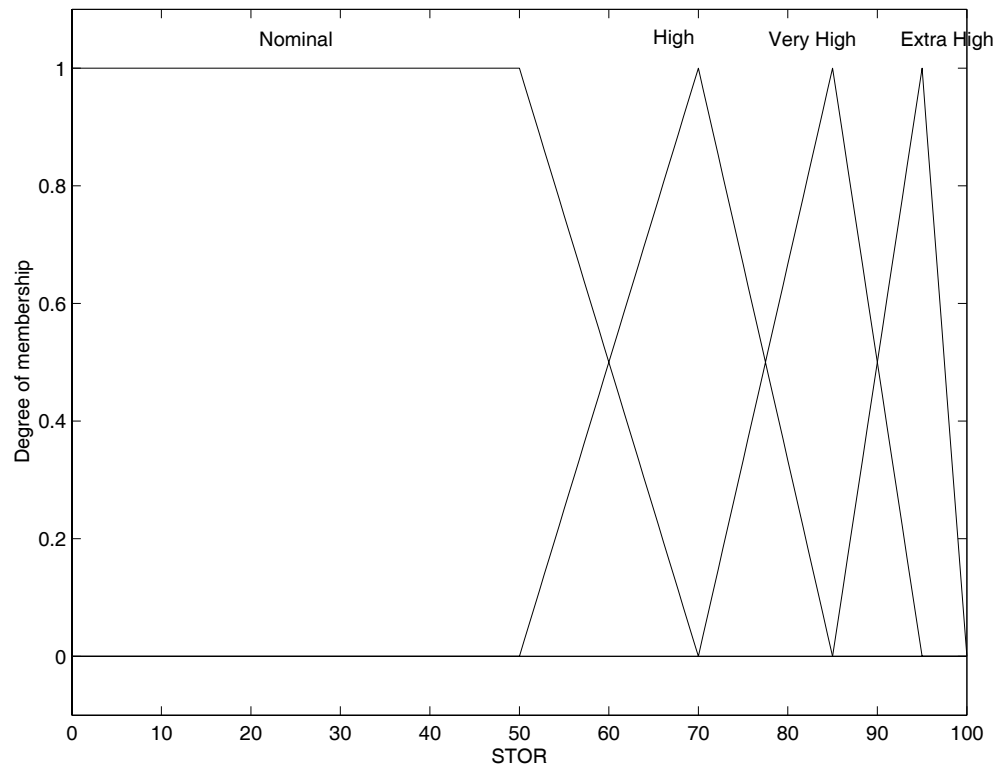
**Figure 60: Consequent MFs for the FIS of TIME Cost driver**

**RULES:**

1. If (TIME is Nominal) then (EFFORT is Unchanged)
2. If (TIME is High) then (EFFORT is Increased)
3. If (TIME is Very High) then (EFFORT is Increased Significantly)
4. If (TIME is Extra High) then (EFFORT is Increased Dramatically)

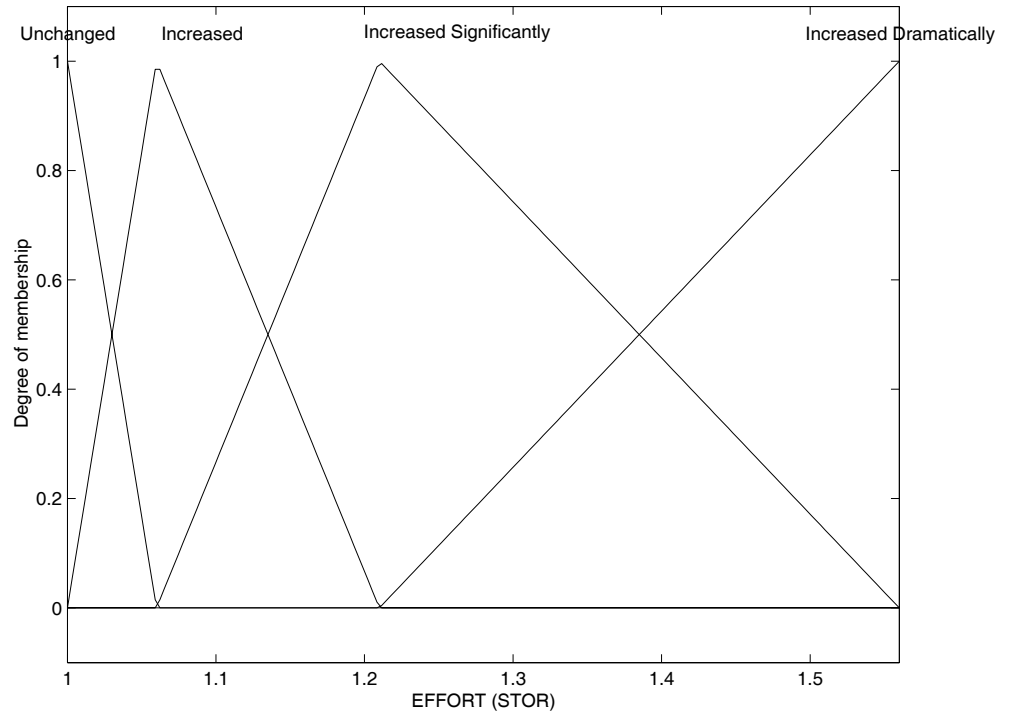
#### 4. STOR – Main Storage Constraint (in percentage)

Nominal	High	Very High	Extra High
$\leq 50\%$	70%	85%	95%



**Figure 61: Antecedent MFs for the FIS of STOR Cost driver**

Nominal	High	Very High	Extra High
1.00	1.06	1.21	1.56



**Figure 62: Consequent MFs for the FIS of STOR Cost driver**

**RULES:**

1. If (STOR is Nominal) then (EFFORT is Unchanged)
2. If (STOR is High) then (EFFORT is Increased)
3. If (STOR is Very High) then (EFFORT is Increased Significantly)
4. If (STOR is Extra High) then (EFFORT is Increased Dramatically)

5. ACAP – Analyst Capability (In percentile)

Very Low	Low	Nominal	High	Very High
15 <sup>th</sup>	35 <sup>th</sup>	55 <sup>th</sup>	75 <sup>th</sup>	90 <sup>th</sup>

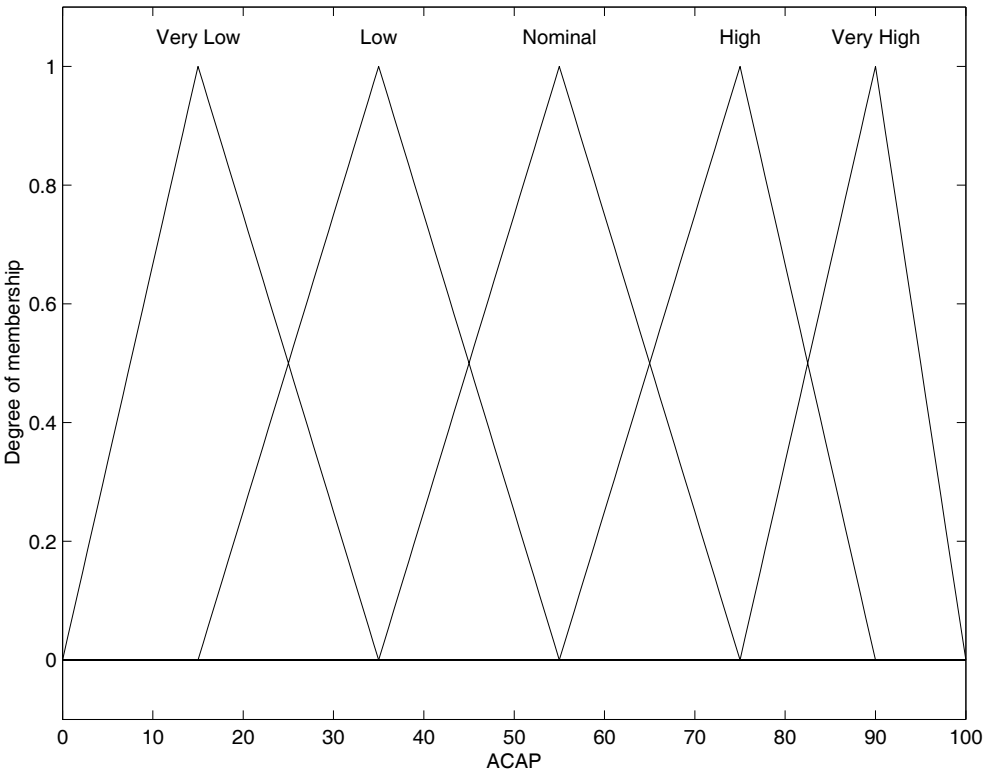
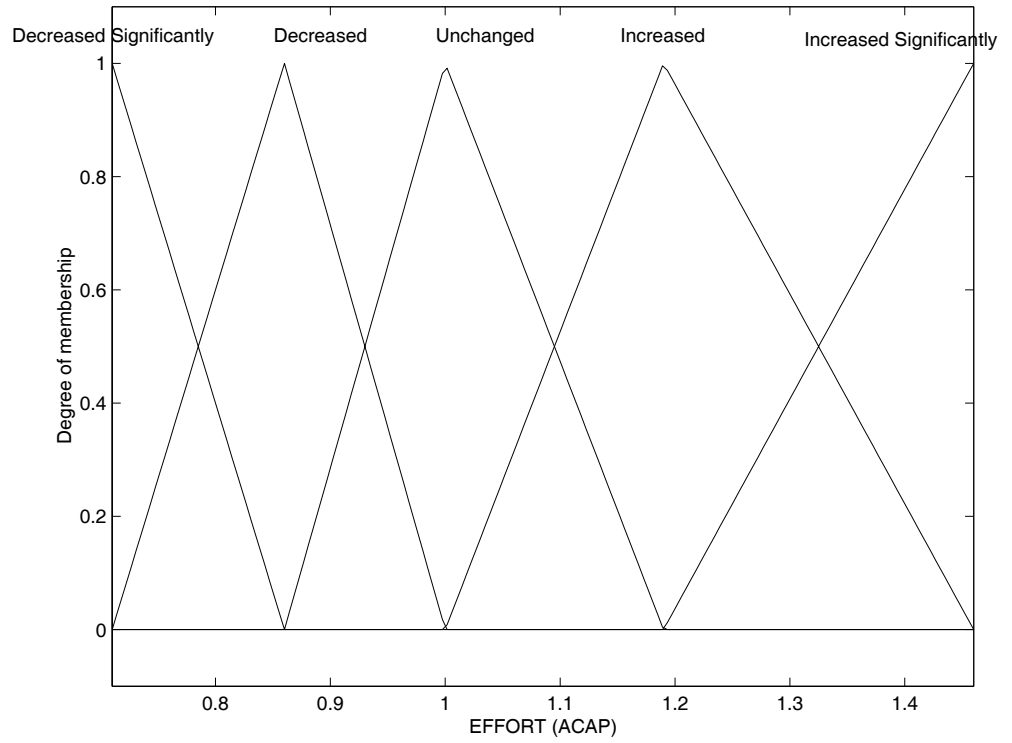


Figure 63: Antecedent MFs for the FIS of ACAP Cost driver

Very Low	Low	Nominal	High	Very High
1.46	1.19	1.00	0.86	0.71



**Figure 64: Consequent MFs for the FIS of ACAP Cost driver**

**RULES:**

1. If (ACAP is Very Low) then (EFFORT is Increased Significantly)
2. If (ACAP is Low) then (EFFORT is Increased)
3. If (ACAP is Nominal) then (EFFORT is Unchanged)
4. If (ACAP is High) then (EFFORT is Decreased)
5. If (ACAP is Very High) then (EFFORT is Decreased Significantly)

6. PCAP – Analyst Capability (In percentile)

Very Low	Low	Nominal	High	Very High
15 <sup>th</sup>	35 <sup>th</sup>	55 <sup>th</sup>	75 <sup>th</sup>	90 <sup>th</sup>

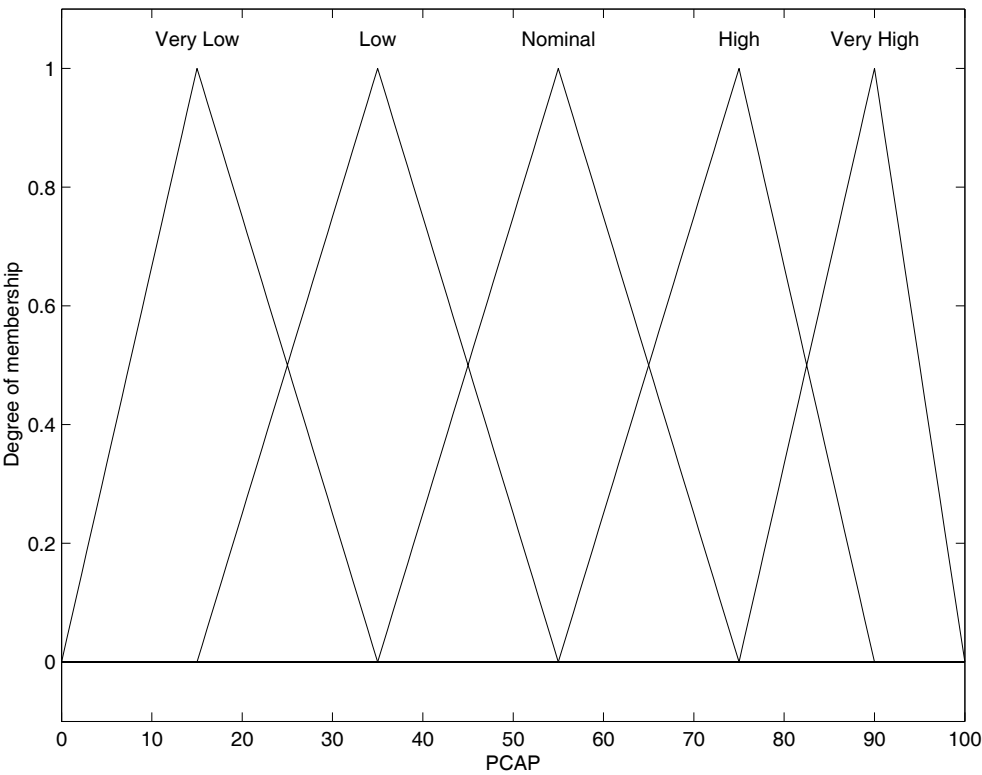
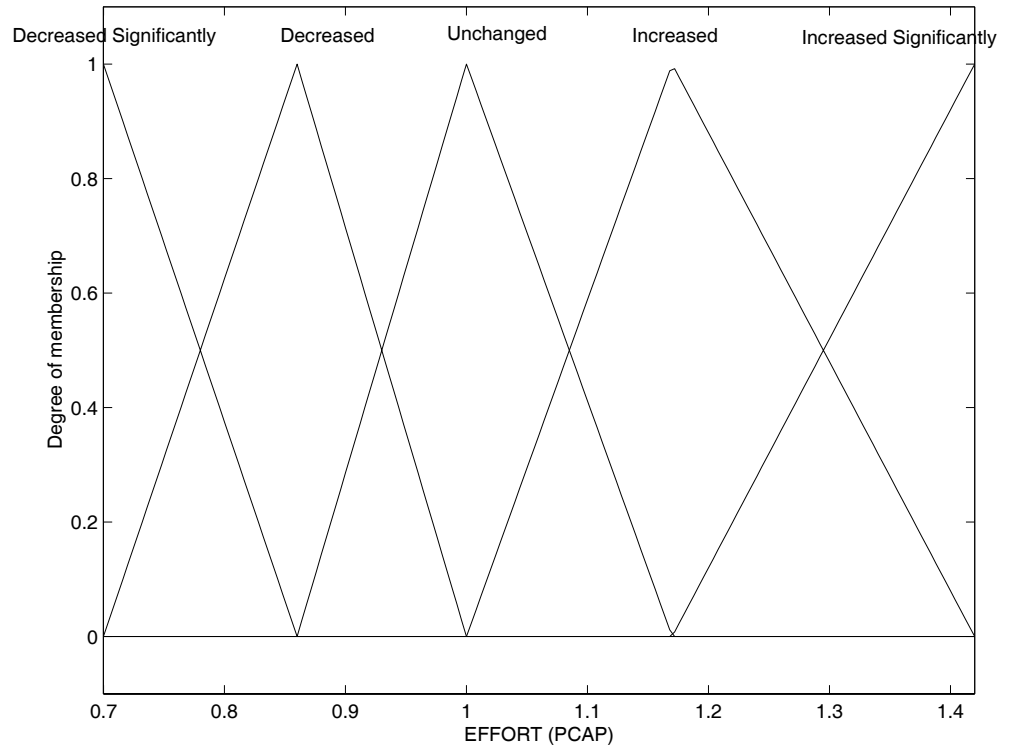


Figure 65: Antecedent MFs for the FIS of PCAP Cost driver

Very Low	Low	Nominal	High	Very High
1.46	1.19	1.00	0.86	0.71



**Figure 66: Consequent MFs for the FIS of PCAP Cost driver**

**RULES:**

1. If (PCAP is Very Low) then (EFFORT is Increased Significantly)
2. If (PCAP is Low) then (EFFORT is Increased)
3. If (PCAP is Nominal) then (EFFORT is Unchanged)
4. If (PCAP is High) then (EFFORT is Decreased)
5. If (PCAP is Very High) then (EFFORT is Decreased Significantly)



7. SCED – Required Development Schedule (Percentage of Nominal Effort)

Very Low	Low	Nominal	High	Very High
75% of Nominal	85%	100%	130%	160%

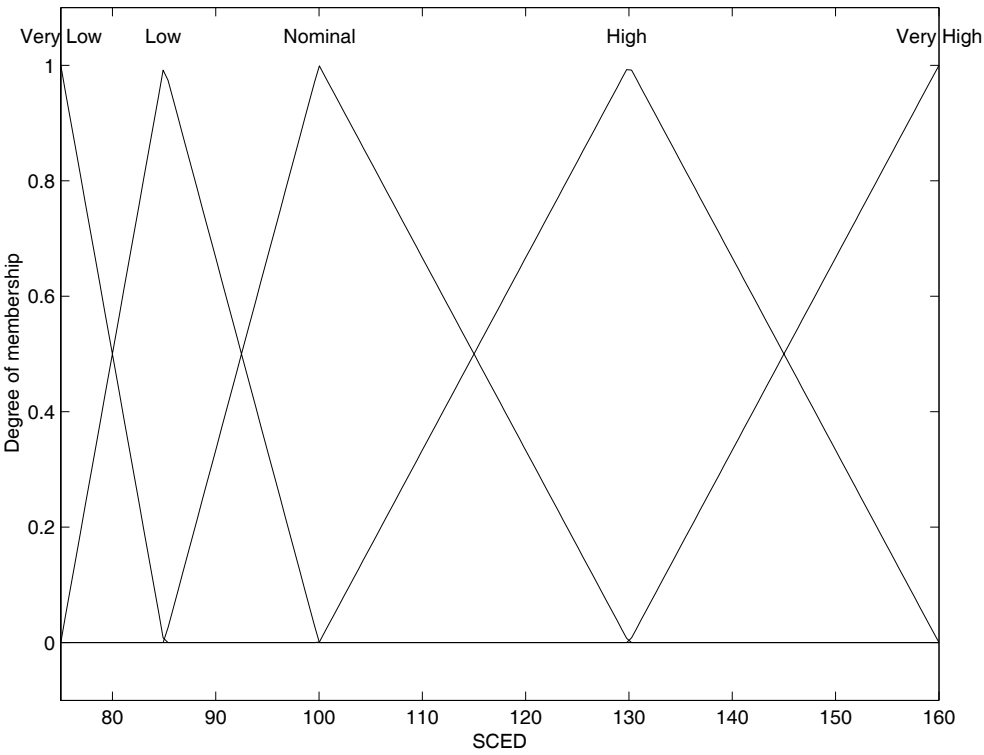
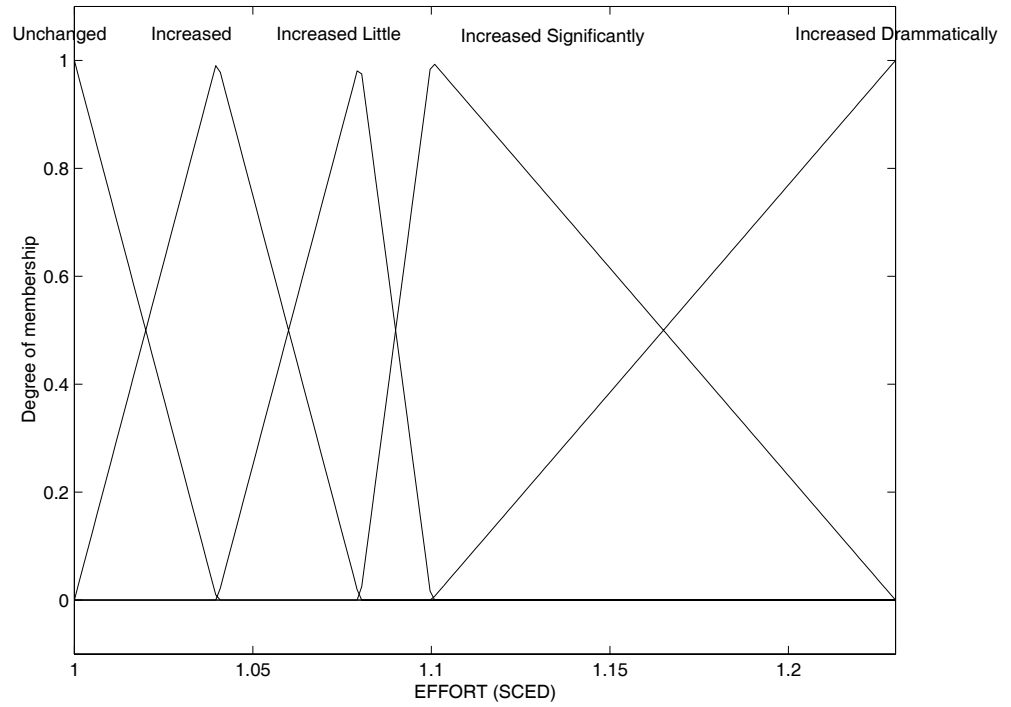


Figure 67: Antecedent MFs for the FIS of SCED Cost driver

Very Low	Low	Nominal	High	Very High
1.23	1.08	1.00	1.04	1.10



**Figure 68: Consequent MFs for the FIS of SCED Cost driver**

**RULES:**

1. If (SCED is Very Low) then (EFFORT is Increased Dramatically)
2. If (SCED is Low) then (EFFORT is Increased Little)
3. If (SCED is Nominal) then (EFFORT is Unchanged)
4. If (SCED is High) then (EFFORT is Increased)
5. If (SCED is Very High) then (EFFORT is Increased Significantly)

8. VIRT – Virtual Machine Volatility (Frequency of change)

Low	Nominal	High	Very High
every 12 months	6 months	2months	2 weeks

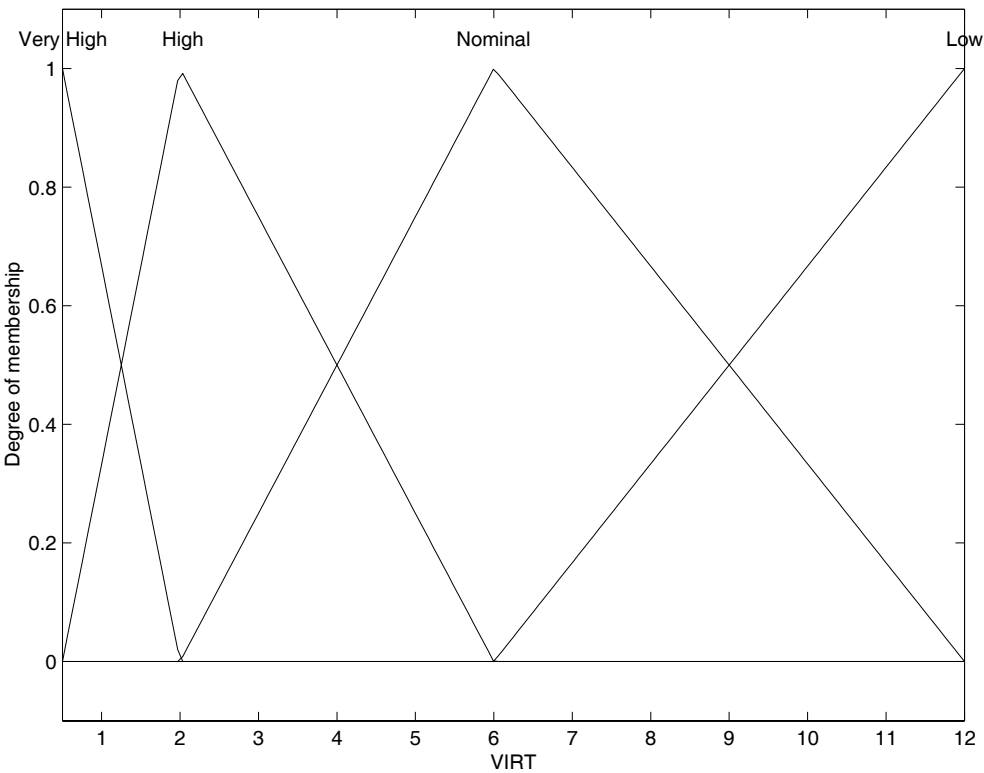
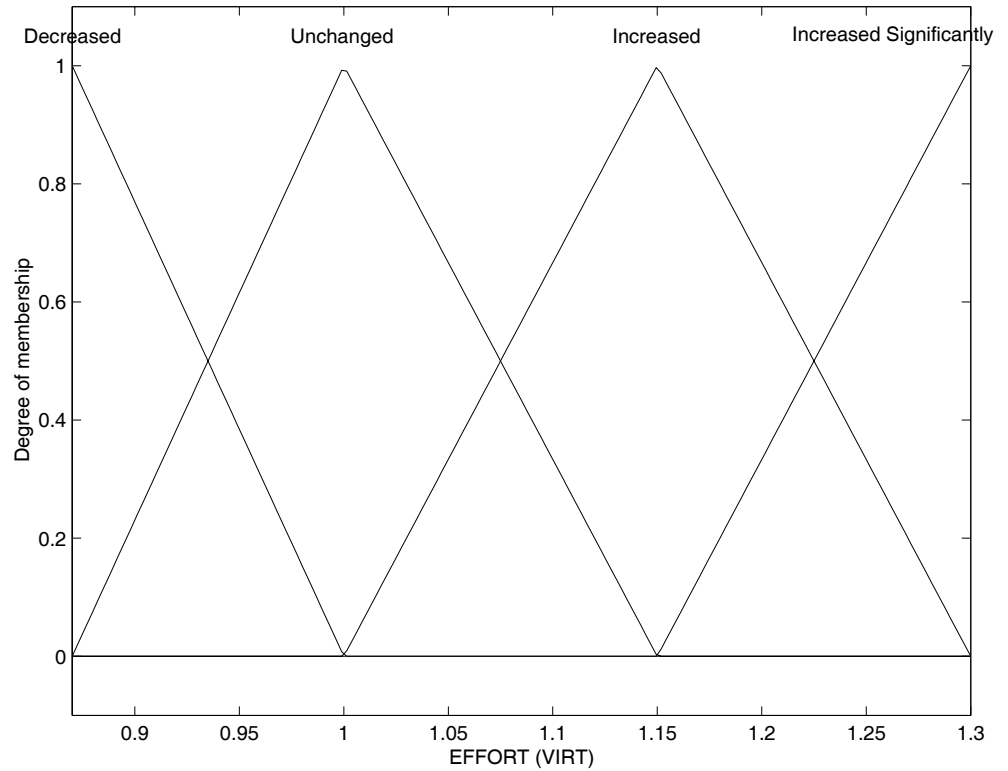


Figure 69: Antecedent MFs for the FIS of VIRT Cost driver

Low	Nominal	High	Very High
0.87	1.00	1.15	1.30



**Figure 70: Consequent MFs for the FIS of VIRT Cost driver**

**RULES:**

1. If (VIRT is Very High) then (EFFORT is Increased Significantly)
2. If (VIRT is High) then (EFFORT is Increased)
3. If (VIRT is Nominal) then (EFFORT is Unchanged)
4. If (VIRT is Low) then (EFFORT is Decreased)

9. AEXP – Applications Experience of the developing team

Very Low	Low	Nominal	High	Very High
≤ 4 months	1 year	3 years	6 years	12 years

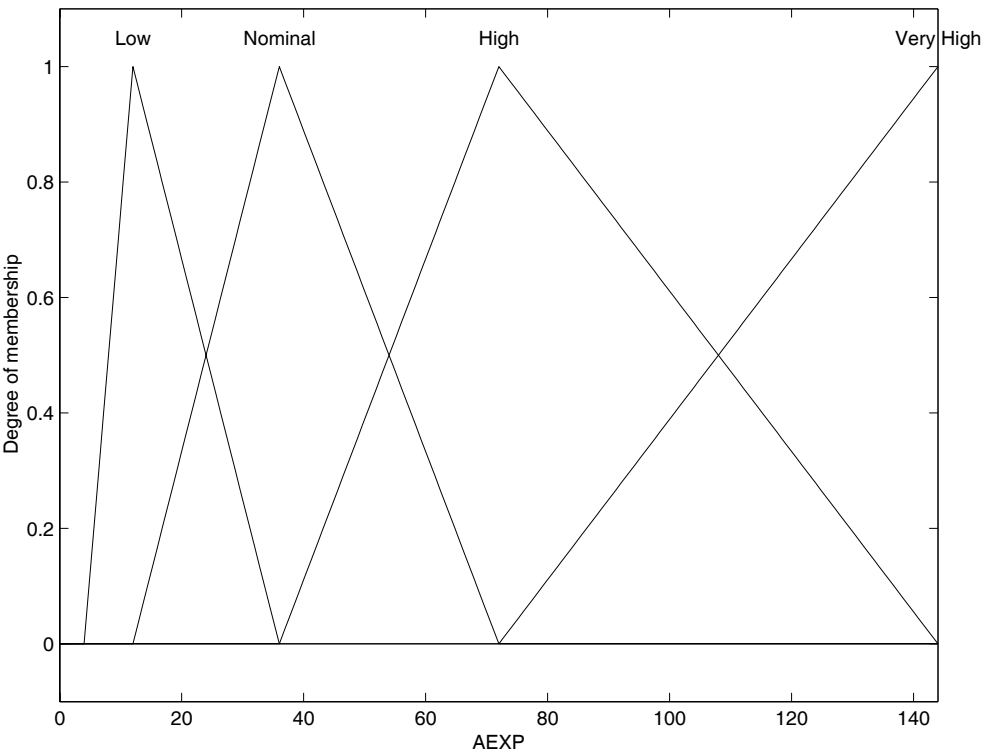
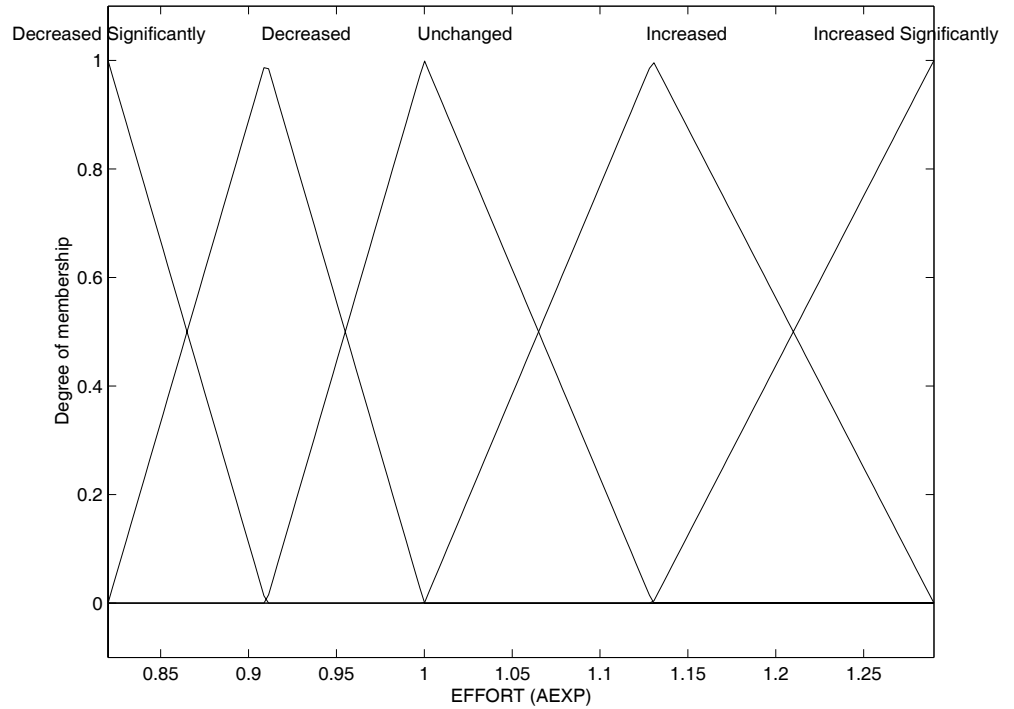


Figure 71: Antecedent MFs for the FIS of AEXP Cost driver

Very Low	Low	Nominal	High	Very High
1.29	1.13	1.00	0.91	0.82



**Figure 72: Consequent MFs for the FIS of AEXP Cost driver**

**RULES:**

1. If (AEXP is Very Low) then (EFFORT is Increased Significantly)
2. If (AEXP is Low) then (EFFORT is Increased)
3. If (AEXP is Nominal) then (EFFORT is Unchanged)
4. If (AEXP is High) then (EFFORT is Decreased)
5. If (AEXP is Very High) then (EFFORT is Decreased Significantly)

10. VEXP – Virtual Machine Experience

Very Low	Low	Nominal	High
≤ 1 month	4 months	1 year	3 years

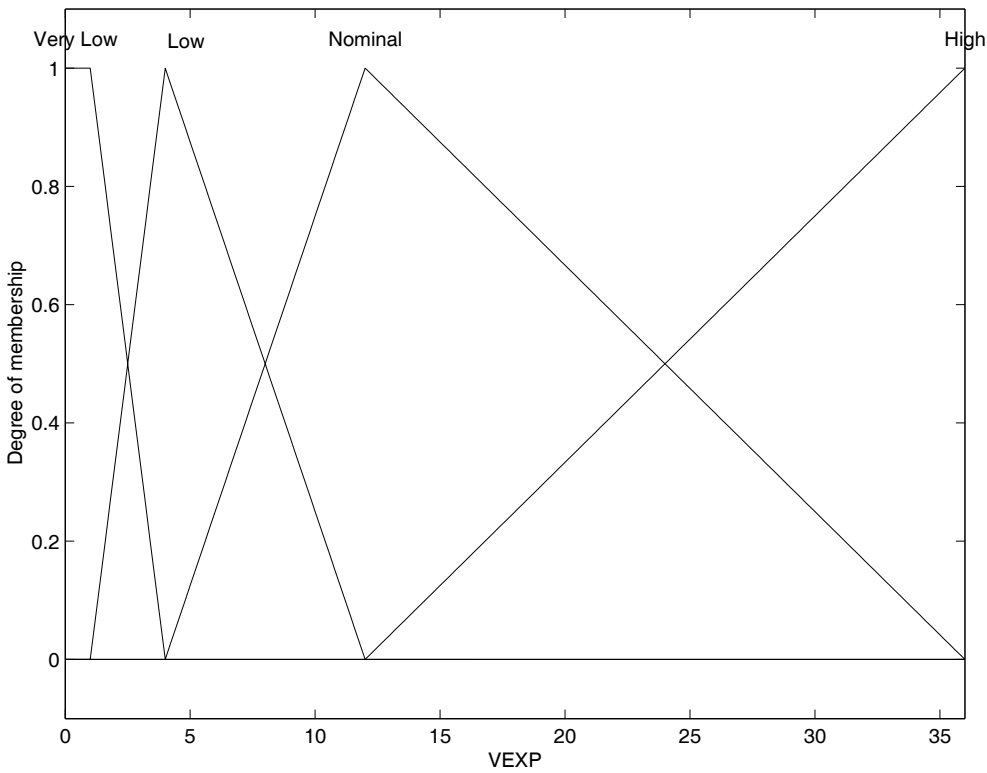
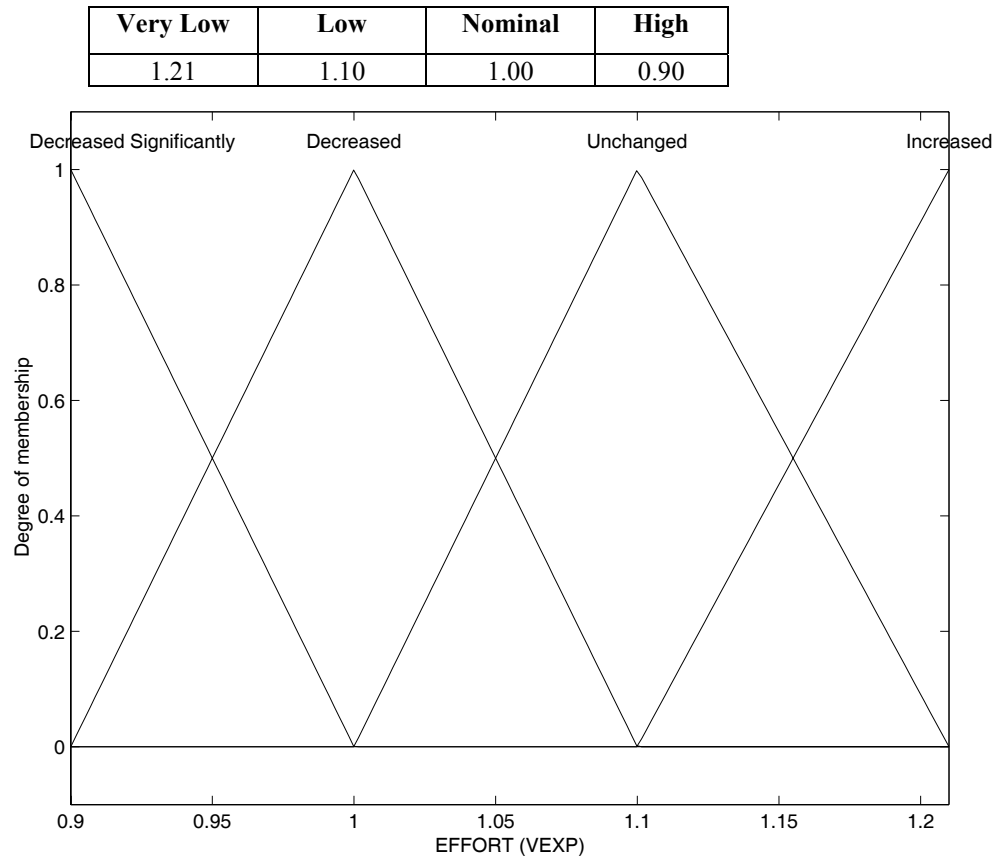


Figure 73: Antecedent MFs for the FIS of VEXP Cost driver



**Figure 74: Consequent MFs for the FIS of VEXP Cost driver**

**RULES:**

1. If (VEXP is Very Low) then (EFFORT is Increased)
2. If (VEXP is Low) then (EFFORT is Unchanged)
3. If (VEXP is Nominal) then (EFFORT is Decreased)
4. If (VEXP is High) then (EFFORT is Decreased Significantly)



11. LEXP – Programming Language Experience

Very Low	Low	Nominal	High
≤ 1 month	4 months	1 year	3 years

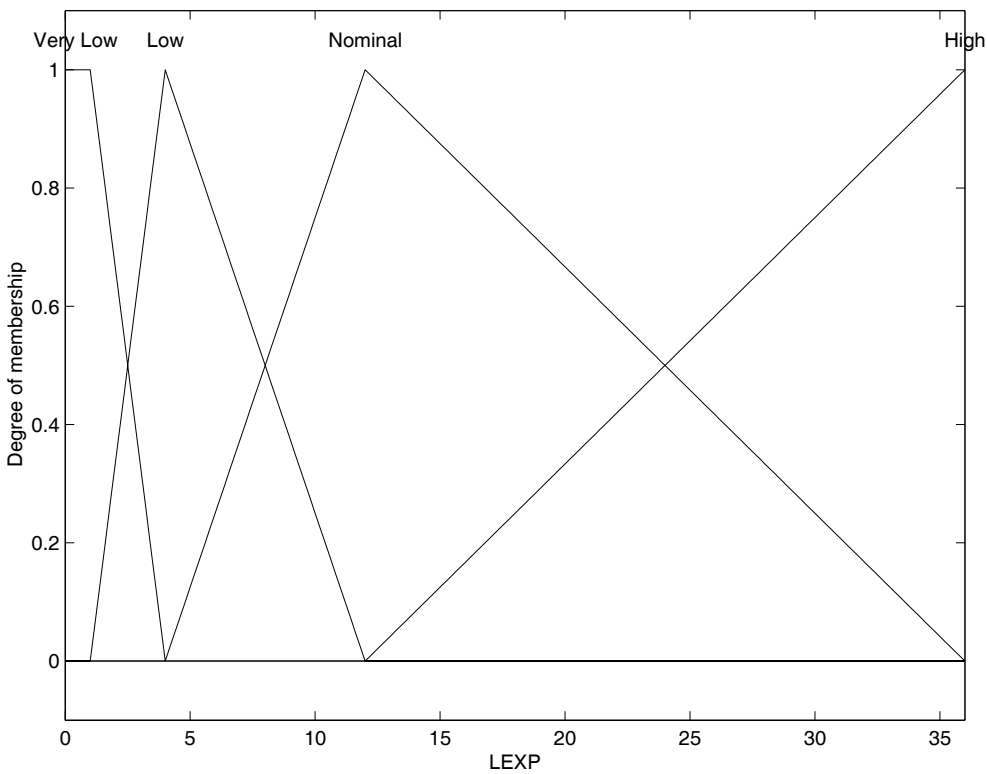
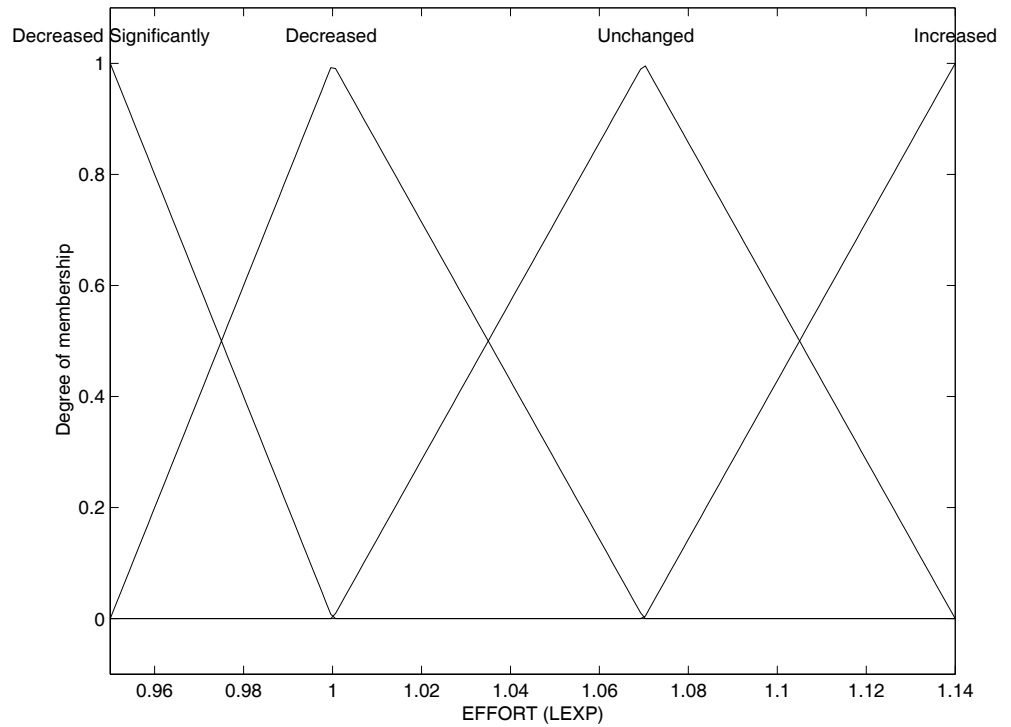


Figure 75: Antecedent MFs for the FIS of LEXP Cost driver

Very Low	Low	Nominal	High
1.14	1.07	1.00	0.95

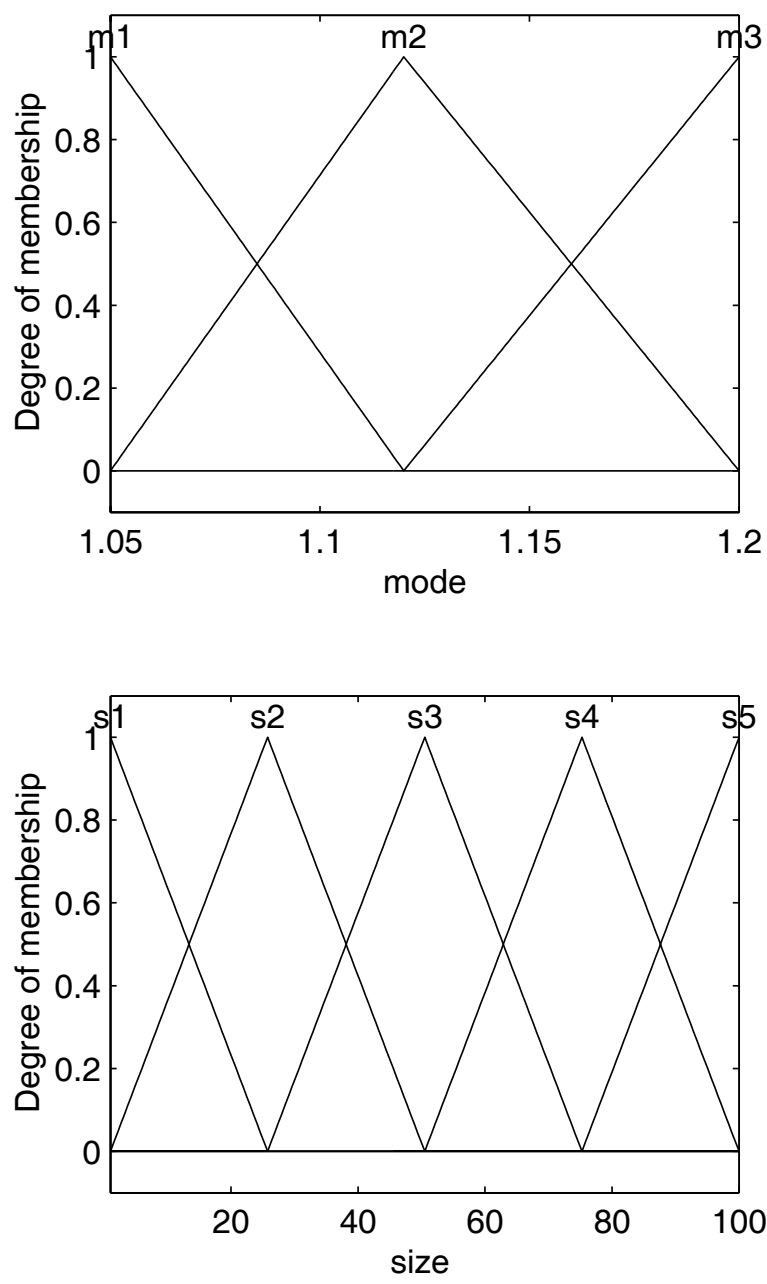


**Figure 76: Consequent MFs for the FIS of LEXP Cost driver**

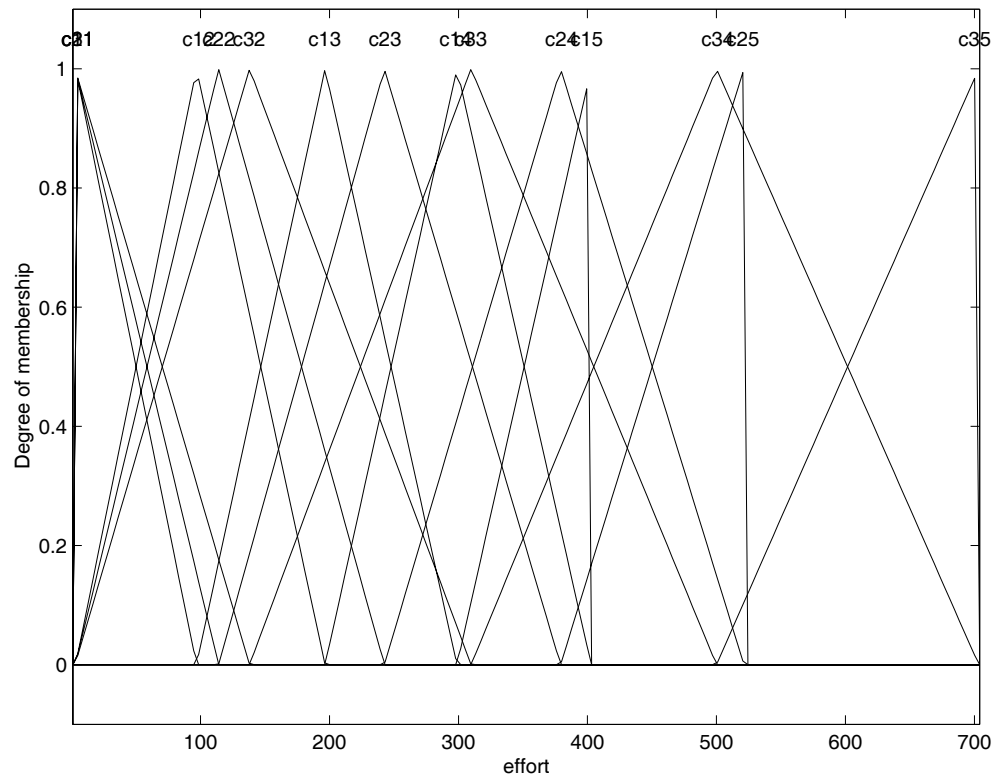
**RULES:**

1. If (LEXP is Very Low) then (EFFORT is Increased)
2. If (LEXP is Low) then (EFFORT is Unchanged)
3. If (LEXP is Nominal) then (EFFORT is Decreased)
4. If (LEXP is High) then (EFFORT is Decreased Significantly)

**Appendix C: SAMPLE FIS**



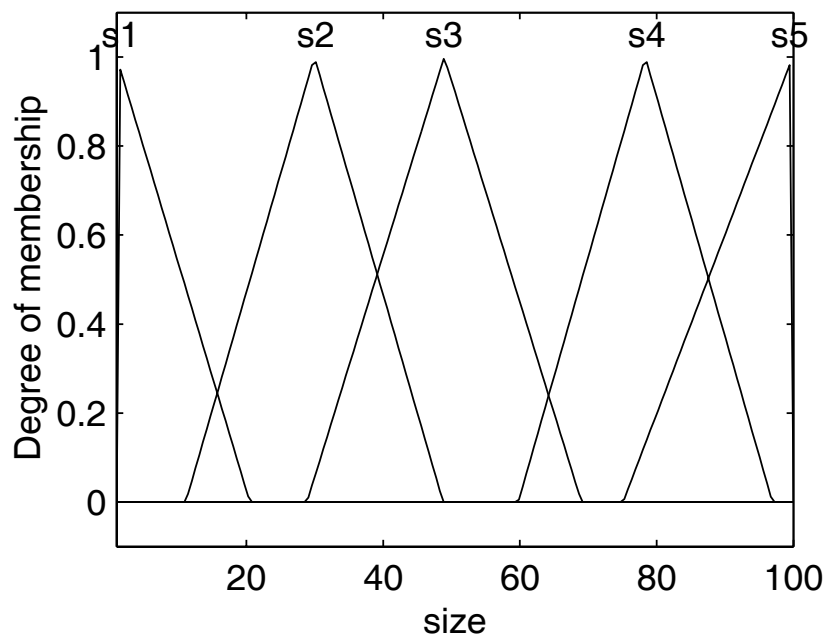
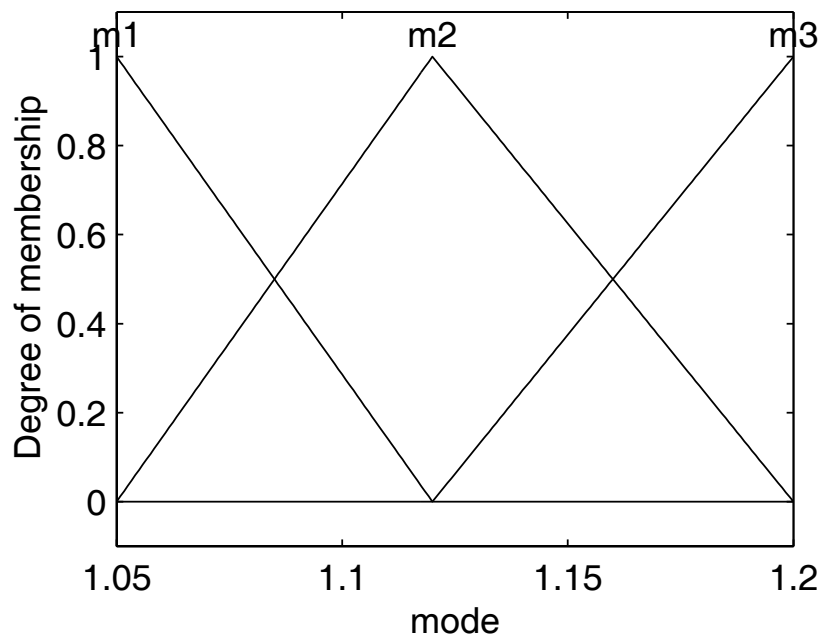
**Figure 77: The Input (Mode and Size) MFs before Training**



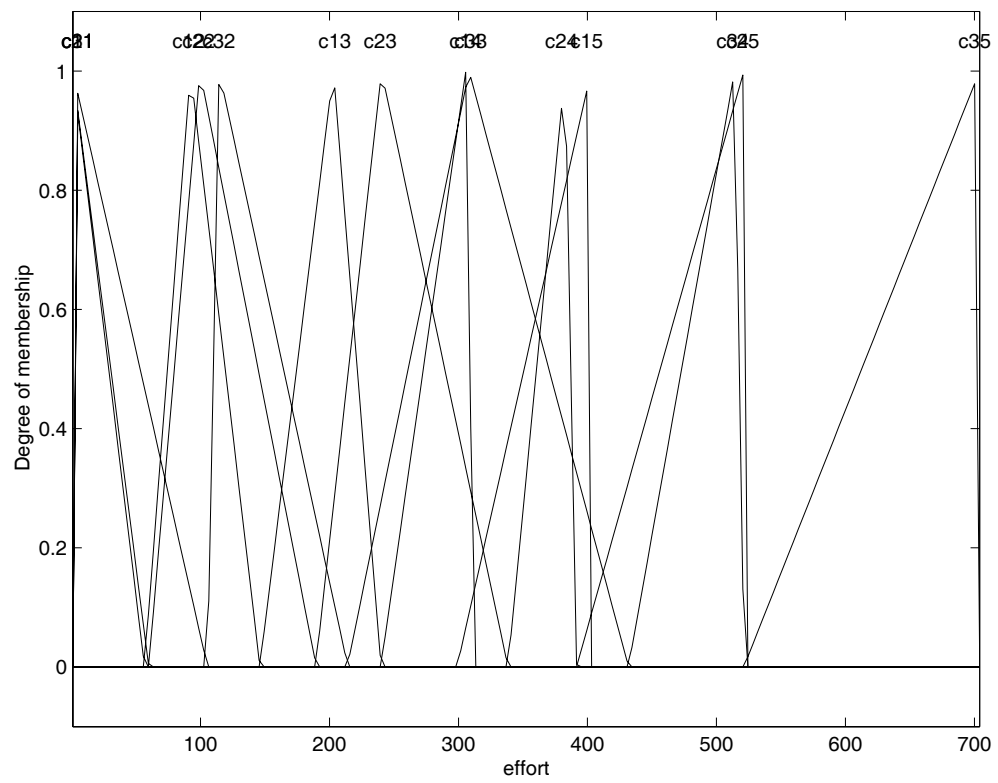
**Figure 78: The Consequent (Effort) MFs before Training**

**NORMAL RULES GENERATED IN THE RULEBASE:**

1. If (mode is m1) and (size is s1) then (effort is c11) (1)
2. If (mode is m2) and (size is s1) then (effort is c21) (1)
3. If (mode is m3) and (size is s1) then (effort is c31) (1)
4. If (mode is m1) and (size is s2) then (effort is c12) (1)
5. If (mode is m2) and (size is s2) then (effort is c22) (1)
6. If (mode is m3) and (size is s2) then (effort is c32) (1)
7. If (mode is m1) and (size is s3) then (effort is c13) (1)
8. If (mode is m2) and (size is s3) then (effort is c23) (1)
9. If (mode is m3) and (size is s3) then (effort is c33) (1)
10. If (mode is m1) and (size is s4) then (effort is c14) (1)
11. If (mode is m2) and (size is s4) then (effort is c24) (1)
12. If (mode is m3) and (size is s4) then (effort is c34) (1)
13. If (mode is m1) and (size is s5) then (effort is c15) (1)
14. If (mode is m2) and (size is s5) then (effort is c25) (1)
15. If (mode is m3) and (size is s5) then (effort is c35) (1)



**Figure 79: The Input (Mode and Size) MFs after Training**



**Figure 80: The Consequent (Effort) MFs after Training**

## **Vita**

Moshood Omolade Saliu, who hails from Ayetoro-Gbede, Nigeria, obtained the Bachelor of Technology (B.Tech.) degree with First Class Honors in Mathematics with Computer Science from Federal University of Technology, Minna, Nigeria in December 1998. Prior to attending King Fahd University of Petroleum & Minerals (KFUPM), he worked as a Programmer/Analyst from May 1999 to January 2001 in SystemSpecs Ltd (a channel partner of Systems Union, UK), Lagos, Nigeria. In January 2001, Omolade joined KFUPM as a Research Assistant to pursue the master's degree. The successful defense of this thesis in April 2003 marks his acquisition of the Master of Science (MS) degree in Computer Science. Omolade's research interests include software metrics, software cost estimation, software process modeling and process improvement, and soft computing. He can be reached at [omolade@omolade.com](mailto:omolade@omolade.com) or [saliu@ieee.org](mailto:saliu@ieee.org).